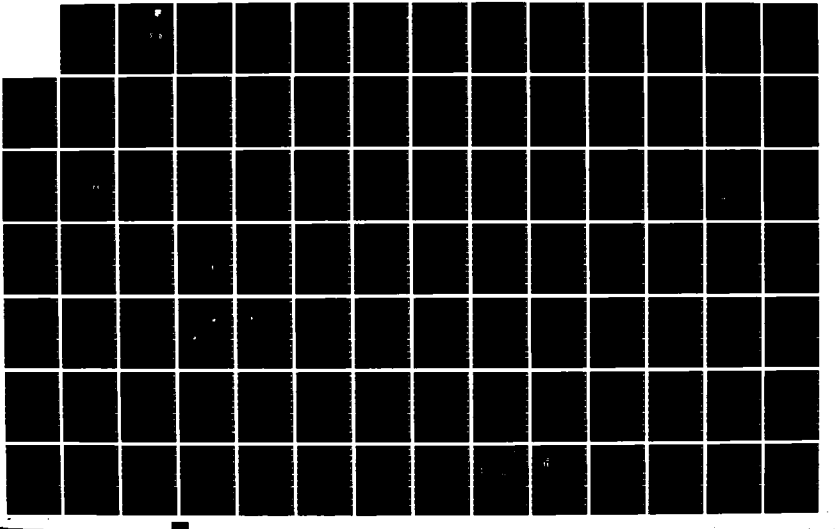
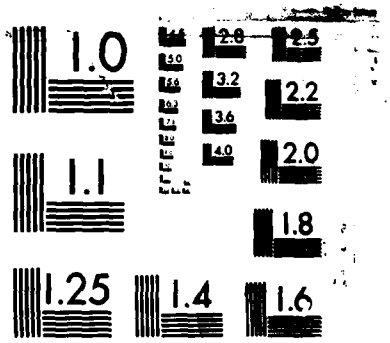
AD-A167 423 C3I RAPID PROTOTYPE INVESTIGATION(U) MARTIN MARIETTA 1/2
AEROSPACE DENVER CO P C DALEY JAN 86 MRC-85-616
RADC-TR-85-216 F30602-83-C-0067

UNCLASSIFIED F/G 17/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS

RADC-TR-85-216
Final Technical Report
January 1986

C³I RAPID PROTOTYPE INVESTIGATION

Martin Marietta Denver Aerospace

Philip C. Daley

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-85-216 has been reviewed and is approved for publication.

APPROVED: William E. Rzepka

WILLIAM E. RZEPKA
Project Engineer

APPROVED: Raymond P. Urtz

RAYMOND P. URTZ, JR.
Technical Director
Command & Control Division

FOR THE COMMANDER: Richard W. Pouliot

RICHARD W. POULIOT
Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

AD-A167423

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS N/A
2a SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) MCR-85-616	5 MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-85-216
6a NAME OF PERFORMING ORGANIZATION Martin Marietta Denver Aerospace 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)
6c ADDRESS (City, State, and ZIP Code) PO Box 179 Denver CO 80201	7b ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a NAME OF FUNDING / SPONSORING ORGANIZATION Rome Air Development Center 8b OFFICE SYMBOL (If applicable) COEE	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-83-C-0067
8c ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62702F; PROJECT NO: 5581; TASK NO: 22; WORK UNIT ACCESSION NO: 12

11 TITLE (Include Security Classification)
C^3I RAPID PROTOTYPE INVESTIGATION

12 PERSONAL AUTHOR(S)
P.C. Daley

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final	FROM Apr 83 TO Jun 85	January 1986	158

16 SUPPLEMENTARY NOTATION
N/A

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP
15	07	
09	02	

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Rapid Prototyping; Development Methodology; C^3I Systems; Modeling; Requirements Analysis

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Rapid prototyping of C^3I systems has high payoff for the Air Force. It can aid in identification and stabilization of requirements, thereby reducing the risk in developing these systems.

Rapid prototyping provides a general tool and way to mock-up the functionality of software intensive systems. The resulting prototype can be exercised before system development to provide a tangible basis and media for presentation of system requirements and design.

Rapid prototyping at this time is best achieved through the use of models of the proposed system's human-computer interface, system performance, data-base structure, and system logical structure.

A system to aid rapid prototyping has been defined. Required attributes of the system are that it be accessible and usable by at least three classes of user: mission user, system acquisition manager, and software specialists.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL William E. Rzepka	22b TELEPHONE (Include Area Code) (315) 330-2762
	22c OFFICE SYMBOL RADC (COEE)

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

i) OUTLINE

1.0 EXECUTIVE SUMMARY

This study was undertaken to assess the feasibility of applying rapid prototyping techniques to Air Force C^3I system developments. This report presents the technical progress during the effort, which entailed studies of problems in developing C^3I systems, approaches to rapid prototyping, relating C^3I system elements to rapid prototyping approaches, and assessing the adequacy of current military standards. During this study we also conducted three demonstrations illustrating prototypical elements of a prototyping system.

Prototyping is a process which has received much recent attention as a way to improve the process of C^3I system development. Rapid prototyping refers to prototyping supported by generalized tools, thereby incurring less effort and time than custom prototyping. By improving the development process through the use of these tools, we expect the system acquisition to be smoother. As an incidental fall-out, we expect the C^3I system product to contain fewer errors and be more usable. Many problems in developing complex C^3I systems are associated with software. Studies have shown that many of the problems of software are tied to requirements definition. This definition process has three aspects (identification, expression, and evaluation) which rapid prototyping must support.

When considering C^3I systems as multi-function systems, we find the portions for which it is most difficult to define requirements are those supporting the cognitive processes of the user. They are also the functions most directly related to the mission of a C^3I system. These are the high-payoff candidates for prototyping. C^3I systems are human systems augmented by ADP.

1.1 Activities Summary

Task One: Investigation

Several large C^3I systems were studied including NORAD and tactical systems. This study included analysis of the acquisition process. A library of over 200 articles and documents was prepared concerning rapid prototyping, software development environments, and structured requirements analysis. An analysis of the problems facing C^3I system developers was undertaken to identify problem drivers. At the same time a study was undertaken of the structure and component functions of C^3I systems and their difficulty of specification.

Current approaches to rapid prototyping were investigated to determine those prototyping approaches which are useful in modeling C^3I embedded computer functions. The cost to use each in terms of hardware, software, analysis, and needed further developments was assessed. Prototyping approaches were investigated which focused on the empirical or visible actions of the software function (clearly important in user interface prototyping), its predicted performance, and its functional structure representation.

A study was undertaken to determine which rapid prototyping techniques are useful in C^3I specific applications. This study addressed the use of parameterized models, reusable software, prefabrication methods, restricted

functionality, and reconfigurable test harnesses. This was performed through review of literature describing the results of prototyping activity and interviews with people who had conducted prototyping for C^3 applications.

The impact which the use of rapid prototyping techniques may have on the Air Force embedded computer acquisition process was studied. This included consideration of what kind of prototyping was appropriate to each phase of the software development life cycle. Modifications to the life cycle were considered as well as the potential schedule impacts of rapid prototyping on the Air Force acquisition process.

An investigation was undertaken into procedures for preserving information gained from prototyping and translating it into requirements/design specifications or the actual implementation. This included cases where the prototype would serve as the program design language, cases where the prototype behavior is examined by the user during the requirements review process, and cases where the prototype is capable of evolving into a final system product.

Task Two: Methodology Development

A method for utilizing rapid prototyping in the development of software for C^3I systems was developed. The methodology was specifically designed for rapid prototyping of high-payoff aspects of C^3I systems to support identification and validation of requirements. The methodology emphasizes the rapid construction, change, and discarding of prototypes instead of evolution of prototypes into final system. Consideration was given to the need for novel approaches to requirements specification, design reviews, and configuration management when using rapid prototyping. The methodology was developed with reference to AFR800-14 and MIL-STD-490.

Task Three: Tool Environment Design

A set of software tools was specified and designed which implement the construction of rapid prototyping consistent with the methodology defined in Task Two above. The tools will form an integrated software prototyping environment with a specialized tool user's interface.

The prototyping system was designed to allow prototype development by users with a variety of skill levels. The tools were designed to support the process of identifying and validating critical C^3I software functions to support the incorporation of the information gained into the decision-making and specification processes which occur during the requirements and design phases of the software development life cycle. This software has been documented in accordance with DOD Standard 7935.

Task Four: Feasibility Demonstration

Three demonstrations were held at Martin Marietta Denver Aerospace to present the prototyping tools and select scenarios. The key demonstration focused on the Ground Attack Control Center (GACC) with a Korean theatre scenario. The intention was to present an element of a GACC as an example of prototyping--not to define or solve a GACC problem. The scenario was incidental. The aspect selected was a user interface prototype of an analyst

performing planning/analysis for a C^3CM mission. Feasibility of the prototyping tools and concepts was demonstrated through identifying and validating some potential user requirements.

1.2 Results Summary

Rapid Prototyping is Feasible with Current Technology. While prototyping has been used for some time in the business world, its application in the crucial arena of C^3 systems has been limited by the complexity of that arena. By applying some of the concepts and lessons from artificial intelligence (specifically planning systems), and by mechanizing the tools using an object oriented executive and tool set, these complexities can be met.

Communication problems between user, acquirer, and developer are the basis for problems in C^3I system developments. These problems destabilize the requirements determination process and in turn leave requirements in a state of flux. Requirements definition for C^3I systems is difficult because of the mission criticality of such systems and because the software of such systems (which provides the prime functionality) must be designed to support human cognitive processing.

Structured analysis and modeling tools are the basis for rapid prototyping of C^3I systems. Rapid prototyping for C^3I systems can best be achieved by binding together powerful modeling tools. The linquistic approach to rapid prototyping is insufficiently mature for application to these systems. The binding process is complicated, and provision for further currently unspecified models in the prototyping system induces a design problem for the prototyping executive.

Prototyping may have an important role as a communications vehicle in multisegment evolutionary acquisitions. While evolutionary acquisition may help in acquiring C^3I systems, organizing large projects is difficult. An example could be taken from the Strategic Defense Initiative (SDI). Prototyping could help determine feasibility early and improve contractor-to-contractor transfer of intentions.

Future C^3I system acquisitions will require new management techniques that must be founded upon tools which explore "what-if" aspects of the problem. With the abandoning of intent to establish a complete requirements set for a system a priori, management techniques must evolve to support a more reactive style. This in turn will call more upon the leadership qualities of program managers and less upon their contingency planning abilities. This results in the need to supply program managers quick reaction tools to give a look into problems at hand and quickly develop likely solutions. This fits nicely into rapid prototyping methodology.

A rapid prototyping environment has major effect when focused on support of requirements determination - not on design. Problems in software intensive developments tend to appear during the software design phase and during the system integration and test phases. These problems are founded in unstable requirements which flow down to the software elements. System requirements determination must be stabilized. Therefore, the proper role for rapid prototyping is in system requirements and high-level software requirements definition. Emphasis should be placed on the integration and test phases in later stages of development.

A rapid prototyping environment should provide that useful work can be performed by a broad spectrum of users with varying skill levels. The proper rapid prototyping environment should, within the tool design itself, accommodate many different users. Each has a piece of the "picture" of the ultimate and proper C^3I system component. The tool should integrate these views automatically to the maximum extent possible. This problem is related to a key artificial intelligence area of research-"knowledge engineering." Complete solution of knowledge engineering problems is not necessary to profit from rapid prototyping, however we must structure the knowledge domain of C^3I system components sufficiently to allow mapping of their specifications to proven representations based in existing modeling systems.

Use of rapid prototyping can take place without disturbing AF800.14 or MIL-STD-490. If we interpret rapid prototyping's main goal as adding to the requirements definition process, then it becomes just another system engineering tool. No special life cycle steps are needed to accommodate use of rapid prototyping.

1.3 Recommendations

A rapid prototyping environment for C^3I systems should be constructed and made available to users, acquirers and developers. The technology is mature and such an environment is feasible. The environment should reside in conjunction with a major Air Force test and evaluation facility focused on C^3I battle management problems because the requirements definition for human cognitive processing aspects of C^3I systems is most intense in the battle management arena.

Provision should be made to interface the rapid prototyping environment to other support environments. These should include programming environments, decision aid development environments, automated management support tools, structured requirements data bases, scenario generators, and further modeling systems. Integration of support tools such as these will give government program managers the capability to play extensive what-if games with the entire gamut of programmatic issues.

2.0 INTRODUCTION

2.1 Purpose

This volume presents our report on the work and results of tasks 4.1.1 through 4.1.4 of contract F30602-83-C-0062.

2.2 Scope

In this volume, we describe our efforts to define, design, and demonstrate a rapid prototyping capability. The capability is being tailored specifically to C^3I system embedded computer functions. This restriction of the problem domain prompts us to use existing technology and reuse existing prototype tool software. Our approach focuses on the rapid construction, use and discarding of prototypes, rather than refining prototypes into final products. We expect the users of the capability to span a range of computer expertise from engineers to mission personnel. Software engineers would construct the prototypes which would be evaluated by the eventual user and possibly modified by acquisition personnel. We discuss our effort to integrate the capability into the Air Force acquisition process. The integration will allow prototyping to improve the validation of C^3I system software early in the development process. Examples, procedures and guidelines have been developed so that all levels of users can understand and apply the capability.

2.3 Tasks

These technical tasks were identified in the contractual statement of work:

2.3.1 Task Technology Investigation

Study at least two large C^3I systems for the purpose of determining those embedded computer functions whose prototypes would have a high payoff in terms of requirement or design information gained. Selected functions should be common to several classes of C^3I systems, critical to system operation and especially difficult to specify and design. Candidate functions include the man-machine interface, real-time interrupt handler, communications processor and the data base management system.

Investigate rapid prototyping technology for the purpose of determining those prototyping approaches which are useful in modeling C^3I embedded computer functions. Hardware and software requirements for each kind of prototyping approach shall be identified along with advantages (kind and quality of information gained) and disadvantages (accuracy, realism and scaling problems). At a minimum, the following three kinds of prototyping shall be studied:

User Interface Prototyping--Captures the user interaction with the system at the expense of system performance and function.

Performance Prototyping--Predicts system time and space characteristics at the expense of user interface and system functionality.

Functional Prototyping--Perform data transformations of the final system without necessarily using the final algorithms, commands or displays.

Study existing rapid prototyping technology for the purpose of determining which rapid prototyping techniques are useful in modeling C^3I functions. At a minimum, the following techniques shall be studied.

Parameterized Models--A family of systems that differ by variations in parameters or tables.

Reusable Software--Libraries of modules which can be quickly and conveniently assembled.

Prefabrication Methods--Programs which accept parameterized inputs and generate specialized functions (e.g., displays, reports) according to some standard.

Restricted Functionality--Model only the functions and only as much of each function as is required to obtain the necessary feedback.

Reconfigurable Test Harness--Simulates the environment of operation and its interaction with the prototype to determine behavior.

Study the impact which the use of rapid prototyping techniques may have on the Air Force embedded computer acquisition process. The kind of prototyping appropriate to each phase of the software development life cycle shall be identified. Additional life cycle steps required by the use of rapid prototyping shall be identified, and a plan for integrating these steps into the Air Force software acquisition process shall be developed. The cost and schedule impacts of using rapid prototyping technology on the Air Force acquisition process shall be investigated, and appropriate management techniques for their assessment and control shall be identified.

Investigate procedures for preserving and translating information gained from prototyping into requirements/design specifications or the actual implementation. At a minimum the following coupling techniques shall be studied:

Feeding Design--The prototype serves as a design specification for the implementation, in effect, being used as the program design language.

Requirements Review and Testing--Prototype behavior is examined by the user during the requirements review process.

Incremental Redevelopment--Under certain circumstances it may be possible to refine selected parts of the prototype into a final system product with the required functionality and performance.

2.3.2 Task Methodology Development

Based on the results of the technology investigation (reference paragraph 4.1.1), the contractor shall develop a methodology for utilizing rapid prototyping technology in the development of software for

C^3I embedded computer systems. The methodology shall incorporate the following concepts, at a minimum:

The methodology shall be specifically designed for the rapid prototyping of critical, high payoff functions of C^3I embedded computer systems. Its objective is to identify and validate these functions during the requirements and design phases of software development. The information gained from the rapid prototyping process shall be used to facilitate the design decision making and specification processes.

The methodology shall emphasize the rapid construction, change and discarding of prototypes, as opposed to a more structured implementation approach which could be refined into the final system.

The methodology shall consider that a different approach to requirements specification, unspecified capabilities, design reviews and configuration management is required when rapid prototyping is utilized.

The methodology shall present to its users a comprehensive set of procedures and guidelines for using rapid prototyping as an integral part of the software development life cycle as specified by Air Force and DOD regulations and standards (reference AFR 800-14 and MIL-STD-490) for acquiring C^3I embedded computer systems.

The methodology shall be fully documented with examples to enhance its understandability.

2.3.3 Task Tool Environment Design:

Design a set of software tools which implement the construction of rapid prototyping consistent with the methodology defined in paragraph 2.3.2. The tools shall form an integrated software prototyping environment with a uniform tool user's interface and shall incorporate the following concepts:

The tools shall be designed for prototyping the critical and high payoff functions of C^3I embedded computer systems. Their design shall be flexible to allow prototype development by technical analysts, prototype modification by Air Force software acquisition specialists and prototype execution by Air Force Mission personnel.

The tools shall be designed to support the process of identifying and validating critical C^3I embedded computer functions and of incorporating the information gained into decision making and specification processes which occur during the requirements and design phases of the software development life cycle.

All newly generated software shall be provided in accordance with paragraphs 5.3.1 and 5.3.3.1 of CP 0787796100e, entitled, "RADC Computer Software Development Specification, General Specification for", dated 30 May 1979. Software documentation will be in accordance with DOD standard 7935.1-S, entitled "Automated Data Systems Documentation Standards", dated 13 Sep 1977 (see CDRL).

2.3.4 Task Feasibility Demonstration

Demonstrate at the contractor's facility the feasibility of the rapid prototyping methodology and tools developed in paragraphs 4.1.2 and 4.1.3 by constructing a user interface prototype of a contractor-selected, Government-approved C^3I embedded computer function. Feasibility shall be shown by using the prototype to assist in identifying and validating user requirements. The demonstration shall also show how information gained from the prototype can be integrated into the decision-making and specification processes.

Utilize existing technology in constructing the demonstration system. Design and implementation of new software shall be held to a minimum and used primarily to interconnect existing software capabilities.

2.3.5 Task Oral Presentations

The contractor shall conduct two oral presentations at his facility. The first, presenting the design of the rapid prototyping tools, shall be held in the twentieth (20th) month after the start of the effort. The second oral presentation shall be held in the twenty-fourth (24th) month after the start of the effort.

3.0 PROBLEMS IN DEVELOPMENT OF C^3I SYSTEMS

3.1 Overview

The development and acquisition of C^3I systems has long been plagued by problems which have limited the effectiveness of those systems when finally fielded. This lack of effectiveness has also broadened the gap between users and developers which leads to an even more difficult development and acquisition process. The purpose of this section is to identify and address these evolving problems, providing the basis for a solution to this growing concern. The following paragraphs provide a definition of C^3I systems to establish the context of the problem, a review of past development and acquisition problems, and finally an analysis of how those problems have evolved to date.

3.2 C^3I System Definition

Before stating the problems involved in the development and acquisition of C^3I systems, it is critical that a definition of those systems be provided. Within the context of this paper, and as defined in JCS Pub. 1, a C^3I system enables the military commander to accomplish the basic functions of command and control (planning, directing, and controlling) in the conduct of military operations. The specific definition of a C^2 system is: "The facilities, equipment, communications, procedures, and personnel essential to a commander for planning, directing, and controlling operations of assigned forces pursuant to the missions assigned". While the functions are fundamental to any and all military environments, the C^3I systems must be structured appropriately for the particular conditions encountered within each environment. Therefore, these types of systems can have numerous complex and changing external and internal interactions, often of an inter-service or multinational (as in NATO) nature. They involve operational requirements, use acceptance criteria and measures of worth which often cannot adequately be specified in advance. They are also highly dependent on the specific doctrine, procedures, threat, geographic constraints, mission scenarios and management approaches of specific mission users. As a result of these considerations, C^3I systems are often subject to frequent change throughout their life cycle. Finally, these systems are normally dominated by software which assists the decision making processes of mission commanders and their staffs at multiple organizational levels.

C^3I systems can be viewed from their component and subcomponent make-up. C^3I systems can be defined in terms of communications, data presentation/ analysis, data management and system management components. Each component can then be further decomposed into subcomponents. Communications incorporates message handling, local area networking, secure communications and sanitization, while data presentation/analysis includes user interface, decision aids, AI techniques and Tracking/Correlation/Fusion. Data management is composed of data base management systems, data structure, concurrency controls, and multi-level secure access. Finally, system management includes resource management and fault tolerance.

The preceding discussion of C^3I systems points out characteristics which lead to the search for improved development and acquisition techniques. Existing techniques are more readily applicable to weapon systems which

generally have discrete, relatively predictable functions, while in C^3I systems changes in requirements are inevitable at some point in the development cycle if the system is to fulfill its primary objective--supporting the human decision maker. In fact, an average of 60% of the cost of C^3I systems occurs post-deployment. This represents changes to the system as the result of changed requirements.

3.3 History

The inadequacy of the traditional development and acquisition approach has long been recognized. There are many examples of cost growth, program delays, equipment deemed obsolete by the time it is fielded, and general user dissatisfaction with systems when they finally are fielded. Figure 3.3-1 shows some of the recent Air Force specific C^2 programs. The Army's Tactical Operations System/Operable Segment (TOS/OS) program, the original version of the Navy's Tactical Flag Command Center (TFCC) and the Air Force Tactical Air Control Center Automation (TACC AUTO) program are but three examples of programs that evidenced these problems and were cancelled as a result.

Program Name	User	Mission	Dates	Success	Program Title
Air Force					
OASIS	USAFE	Special Intelligence Mgmt	78-85	S	Operational Application of Special Intelligence Systems
427M	NORAD		70-79	PF	-
NCMC Upgrade	NORAD	I&W and Defense Mgmt	79-83	S	NORAD Cheyenne Mountain Center Upgrade
TACC AUTO	TAC	Tactical Battle Mgmt	67-80	F	Tactical Air Control Center Automation
CAFMS	9th, 12th TAF	↓	79-81	S	Computer-Aided Force Management System
EIFEL I	NATO RSI	↓	77-82	PS	-
Constant Watch	PACAF	↓	75-	PS	-
ENSCE	TAC	Analysis/Fusion	81-	PS	Enemy Situation Correlation Element
Joint					
BETA	Army/TAC	Analysis/Fusion	77-81	PF	Battlefield Exploitation and Target Acquisition
Joint Tactical Fusion Program	Army/TAC	Analysis/Fusion	81-	PS	Joint Tactical Fusion Project

Legend:

S Successful; PS Partially Successful; PF Partial Failure; F Failure

Figure 3.3-1 Some Air Force Command and Control System Acquisitions

As a result, the Air Force user community, under the direction of the Air Staff, has taken on the acquisition role for several of their own C^3I programs. Examples include PACAF's Constant Watch Program, TAC's Computer Aided Force Management System (CAFMS) and USAFE's EIFEL I System. This activity further complicates the problems with C^3I system development and acquisition with the costly duplication of acquisition efforts at each of the operating commands.

In general, the many problems associated with C^3I development and acquisition stem from a long history of "force fitting" C^3I systems into weapon system molds. Although the fundamental problems have been recognized, few coordinated efforts have been made to improve the situation. Therefore, the problems of the past are the problems of the present.

3.4 Problems

Primarily there are four areas of potential concern in approaching acquisition and development problems associated with C^3I. These general areas can be categorized as: Technical, Communications, Cost, and Scheduling. The last two areas, cost and scheduling, are normally driven by technical and sociological factors in similar manners. In today's environment of rapid technical advances and capability improvements, technical problems such as wrong ADP choice does not adversely impact C^3I system acquisition and development as it has in the past.

Most of the major problems facing the C^3I community in today's environment fall into the communications category. C^3I development and acquisition requires complex, multi-faceted interactions that occur throughout the life cycle of the development and acquisition phases within and among all members of the community including the user, the acquisition agencies and the developers as shown in Figure 3.4-1. However, C^3I systems have been forced into the mold of weapon system development and acquisition techniques which have proved inefficient due to the one-of-a-kind nature of the systems. The "force fitting" has prevented many of the essential human interactions from taking place. These interactions must occur at all stages and levels during the development of any system designed to aid in decision making and execution. All too often, this interaction has been lacking in past systems designed for this purpose and the end results were systems that did not "do" what the user expected.

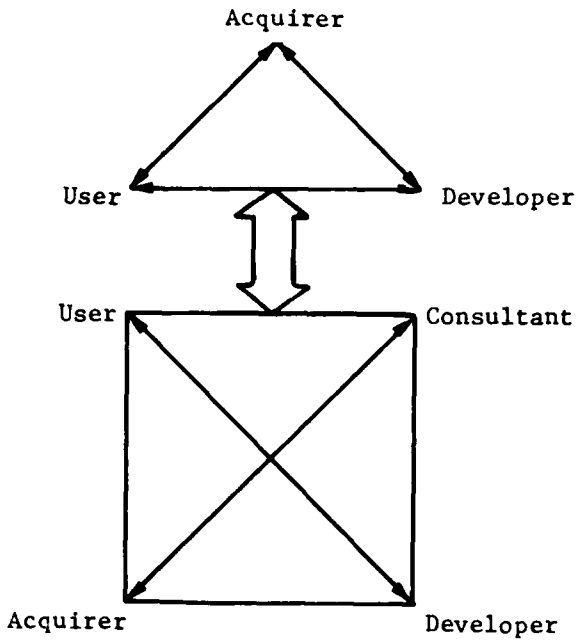


Figure 3.4-1
Communications paths within current acquisition can be complex.

Lack of communications between the user and the acquisition and development agencies has resulted in systems which are not interoperable, utilizing incompatible hardware and/or software designs. They also often lack in their ability for growth and refinement. Therefore, C^3I systems designed and developed in isolation may not provide the capabilities required by the user.

Escalating costs in system development and acquisition continues to be a major concern. Beside being driven by problems created by technology and sociology, other factors such as funding strategy changes (both politically based and internal to the DoD) have major impacts on the fielding of new systems. Likewise, social and technical problems create problems in meeting schedules. At present, the time required from the program start to the final realization of a fielded C^3I system can normally be measured in years. We define the program start as the initial activity for a specific program such as creation of an SPO, directive of program start, or approval of an SON. A study [Affordable Acquisition Approaches, A3, 1980] for Commander AFSC analyzing trends in Air Force system acquisition has projected 10 years as the average development time for a C^3 system when measured from program start through start of full scale development (DSARC II, PMD directing FSD or initial 6.4 funds expenditure) to first production delivery.

Large-Scale C^3I Developments

C^3I system developments cannot be viewed in isolation. The nation's command and control system exists in what might be viewed as large chunks. A problem facing program managers on both the part of government and industry is planning within multi-segment upgrades. Consider the Space Defense Command and Control System (SPADCCS). Elements of the SPADCCS upgrade include sensor improvements, communications system improvements, combined Space Operations Center (CSOC), SPADOC 4, IDHS upgrade, CSSR, and SPADCCS integration contract. These are all occurring simultaneously and all are using the evolutionary acquisition approach. Two problem areas exist. The first is functional prioritization within a segment. What are the desired functions/capabilities of the segment? Which of these are the core required capabilities? How should the capability list be time-ordered to facilitate EA block planning? By exposing some of the deeper areas in the requirements set through prototyping, information necessary to answering the questions may be obtained.

The second problem area is in aligning required performance across segments to facilitate overall system integration and uniform performance improvements as shown in Figure 3.4-2. When must a particular block be finished? Are the block phasings across all segments consistent? We are pointing out that in large multi-segment C^3I system acquisitions, there is an additional destabilizing factor provided by the existence of parallel developments. While the evolutionary approach mitigates some of this influence, there remains a complex effort to provide the planning which allows the evolutionary approach to realize its potential. Prototyping can aid in resolving this problem through provision of a communications medium and basis across segment participants.

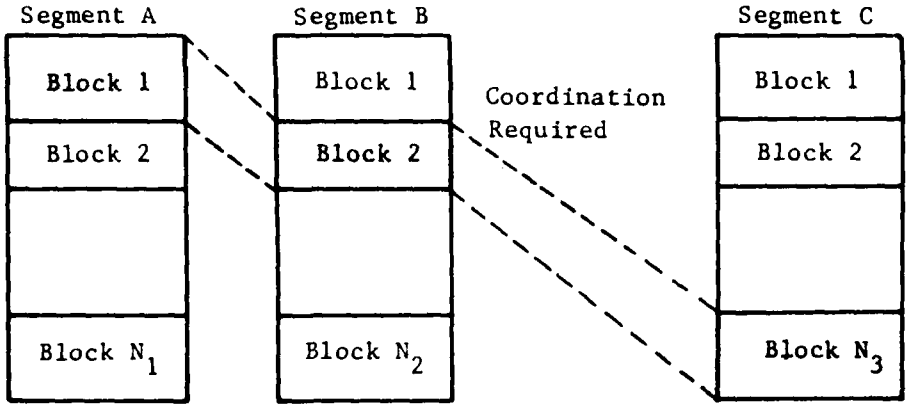


Figure 3.4-2

Large-scale C^3I system present further problems such as coordinating block contents and deliveries across several segments.

3.5 Taxonomy of Errors

We have prepared a taxonomy of the types of errors which adversely impact C^3I development. This is not a usual tree structured taxonomy in that all nodes separated by a plus sign may be active at the same time. That is, there may be many different error types occurring on a program simultaneously. This structure is shown below in Figure 3.5-1.

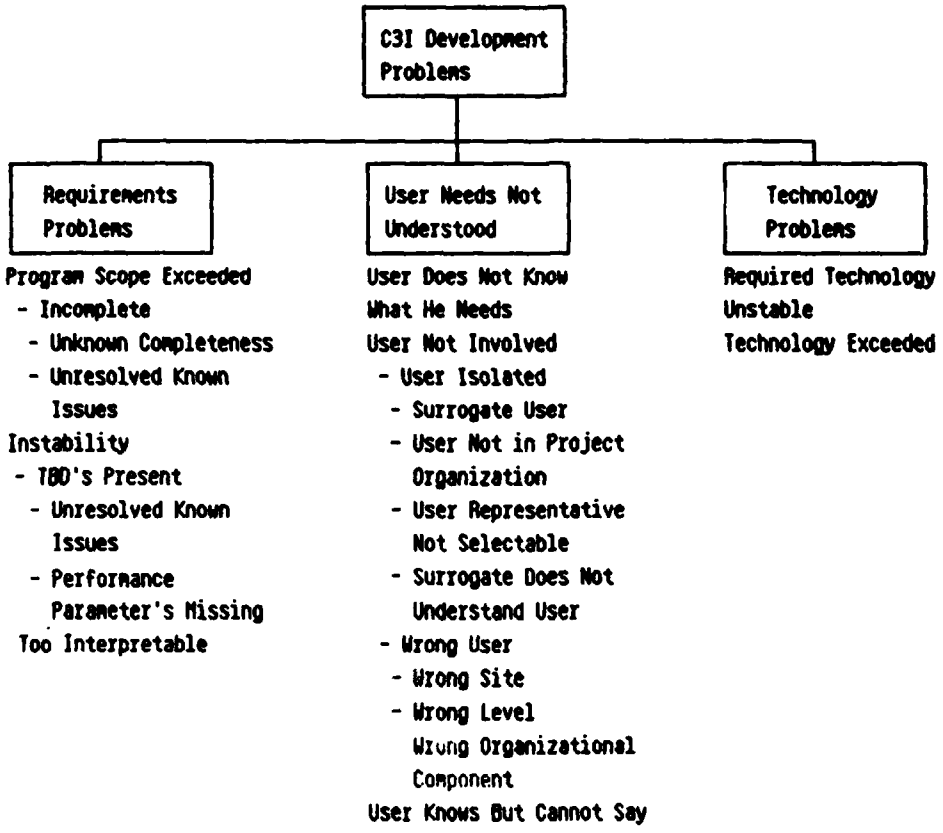


Figure 3.5-1 Problem Taxonomy

PROBLEM TAXONOMY

This taxonomy (Fig. 3.5-1) is the current perception of where problems arise in C^3I system developments.

Level 0 C^3I Development Problems =
(Requirements Problems + User Needs Not Understood + Technology Problems)

Level 1 Requirements Problems =
(Program Scope Exceeded + Instability + Too Interpretable)

Level 2 Instability =
(Incomplete + TBDs Present)

Level 3 TBDs Present =
(Performance Parameters Missing + Unresolved Known Issues)

Level 3 Incomplete =
(Unknown Incompleteness + Unresolved Known Issues)

Level 2 User Needs Not Understood =
(User Not Involved + User Does Not Know What He Needs + User Knows But Cannot Say)

Level 3 User Not Involved =
(User Isolated + Wrong User)

Level 4 Wrong User =
(Wrong Site + Wrong Level + Wrong Organizational Component)

Level 4 User Isolated =
(User Not In Project Organization, Surrogate User)

Level 5 Surrogate User =
(User Representative Not Selectable + Surrogate Does Not Understand User)

Level 1 Technology Problems =
(Required Technology Unstable + Technology Exceeded)

Technology Problems

Some development problems run afoul of the state-of-the-art in technology. The technology necessary to build a particular system may be unstable. An example is memory-staging schemes for high speed parallel processors such as the CDC 205. This error type occurs when state-of-the-art solutions are being used in new applications.

The other branch under technology problems is labeled "technology exceeded." This refers to a development necessitating a solution which is beyond the state-of-the-art. An example is a computer with a sustained throughput rate of 1 GFLOP/sec.

Requirements Problems

These error types appear on the left-hand side of the Figure. At the first level is the node "program scope exceeded." This refers to a requirements set which defines a set of solutions taking more time or money than is available in allocated development funds. The requirements may be good in all other respects.

Another type of error is "unclear, too interpretable." This refers to a requirements set which does not adequately define the task or language which is too loose. An example would be language which said "data base updates shall occur instantaneously".

The third node at this decomposition level is instability of requirements. This has a substructure containing incompleteness and TBDs (to be dones). The most usual TBDs seen in specifications are undetermined performance parameters. It is conceivable that there would be TBDs which are not performance related, such as determining the number of workstations necessary for a site. Incompleteness is in one case equivalent to "other TBDs." This is where some portion of the specification has been overlooked as to number, quality, or other attributes. In the other case, the specification is incomplete and does not adequately define the work to be performed.

User Needs Not Understood

The last major type source of error is under the heading "User needs not understood." This is the classical way requirements analyses go astray.

The current acquisition plan in use by the Air Force can keep the eventual user of the C^3I system insufficiently exposed to the development process. This occurs through isolating the future site user or by involving a wrong/inappropriate user representative. The wrong user representative can occur three ways. The first is through selection of someone familiar with another site in the same command. The second is through selecting someone at the wrong command level. This occurs a great deal. Our studies have shown that there are two user representatives, one at the analyst level and the second at the staff or CO level. Selection of one of these users to the exclusion of the other can bias the determination of user needs. The third way to make an inappropriate selection is to select someone familiar with the site but in the wrong organizational component. An example is found in one of the SAC command center upgrades. There a user representative was chosen from SI, the ADP governing organization. The people whom the system was actually to support were in DC, the command and control organization.

Isolation of the user can come about in two ways. The first is through insufficient integration of the user (user representative) within the developer's organization. The normal project review process is insufficient to provide guidance on complex C^3I systems. If the user is kept at arm's length for the requirements and design phase, his insights, viewpoints, and goals will likely go unheard.

The second way to isolate the user is through the surrogate user system. Various acquiring authorities of the government or consulting companies have been "tagged" as the user representative. While this has worked in some cases, the system introduces further organizational and communications problems. These occur, first, because a representative cannot be found. This was the case on the original Navy TFCC program. No single person or group could be found to integrate the needs of the many flag commanders who would use the system. Their views were not "representable."

A more common occurrence is when the appointed user does not understand the operational environment which the system is supposed to manage. A classic example is on the OASIS program. There a site user desired "annotation" data to be included on displays by the operator. The surrogate user generalized this to "amplification" data, using arbitrary data base sections. The proposed solution went through several levels of project review by all parties, but the disparity was not discovered until the system was delivered to the site user.

3.6 Summary of Problems in Developing C3I Systems

Organizational communication paths are complex. When we have an acquisition that comprises a developer, an acquirer and a user, there are difficulties of translation. When we add a consultant, the number of communication paths proliferates and translation problems multiply. In multi-segment acquisitions we have a very confusing mix of contractors, acquiring authorities or authority, and one or more consultants.

Requirements analysis is notoriously difficult for C^3I systems. Requirements analysis tends to be a well-defined process for systems which we understand and have developed successfully in the past. The threat and its countermeasures are becoming more sophisticated and more capable. The ability of the threat to posture its forces ever closer to the shores of the United States and to have at its disposal long-range aviation assets is challenging. Clearly this results in a reduction of the time available to assess, decide and respond correctly. This in turn drives us to make more demands for accuracy, timeliness and precision on the part of our nation's command and control systems. This forces a closer and more intimate integration of these command and control systems with the user and his cognitive and inferencing processes. This is a new field. We do not have paradigms. We do not have a long, well-documented track record. We are exploring new territory. The problem is one of supporting identification of requirements, expressing and representing those requirements accurately, and evaluating them in a manner so as to provide structured feedback to the identification process. Anything which helps mitigate problems in identification of requirements would be a great boon to our nation's command and control acquirers, developers, users, and consultants.

Acquisition

Development strategies of C^3I systems have proven inappropriate. Standard acquisition strategy is serial and is focused on standard weapon system acquisition descriptions and guiding documents. Such acquisitions provide feedback to the user, basically through document such as those delivered as requirements documents, design documents, and test plans. The validation of requirements at the system level and correspondingly at subordinate levels occurs with the delivery of a system to the hands of the user. There is well-documented lack of success for standard serial acquisitions such as the Tactical Flag Command Center (TFCC) and the Tactical Air Control Center Automation Program (TACCAUTO). The serial nature of this acquisition strategy calls for requirements to be completed for the entire program and then the contractor stops. Design for the entire program commences and then stops. Development occurs, and then integration and test. This presumes that we understand the goals of the program sufficiently when we start to preplan. This does not work for command and control systems.

This has caused many contractors and acquiring personnel to look towards an evolutionary acquisition strategy. An evolutionary acquisition strategy tends to be a block-structured serial acquisition emphasizing many life cycles of requirements design, development, and test. There are mini contracts issued at the beginning of the next block and there is some preliminary analysis done to assure that a basic design is achieved with the first delivery. This has been used successfully on some command and control systems such as Operational Application of Special Intelligence System (OASIS) and the Space Defense Operation Center (SPADOC). The feedback through the user is through documents and the block deliverables. The final delivery is broken up into a staggered set of deliveries of fully functional system versions. There are several requirements validations occurring spread out in time, and this provides the user the ability to feedback to the developer and acquirer crucial data. That which has been left out or done incorrectly due to misunderstanding can be corrected to some extent in the following acquisitions. There is a well documented success for this from the OASIS program, which was the first planned evolutionary acquisition by the Air Force.

The third approach is what we refer to as an experimental or evolutionary approach. It tends to be a free form evolutionary acquisition stressing continuing development. It has been used extensively in the R&D world, and most are familiar with it. The experimental approach calls for work and requirements to progress simultaneously in a continuous and open-ended manner until 1) funds run out, 2) time of the program is exhausted, 3) the technical and/or contractor people give up because they are tired, or 4) they achieve an acceptable level of functionality and therefore success. It tends to be used on well defined subproblems that have embedded kernels whose solutions are unknown. The work on the Navy Secure Message Processor by NRL is one example of the application of this experimental acquisition strategy. Feedback is provided through an almost continuous hands-on access to the system as it is brought into being by the user. Requirements, therefore, are validated almost constantly. Some success in small programs has been achieved. Drawbacks are that these acquisitions are hard to manage, require specialized talent and support environments, and have yet to prove themselves on large acquisitions.

The last acquisition strategy may be termed "opportunistic." It is a combination of the serial and experimental life cycles. There has been some use of it in R&D work. The opportunistic approach calls for simultaneous, parallel progress of standard serial acquisition and experimental acquisition. The experimental acquisition is a prototyping activity feeding the requirements and design activities of the serial acquisition as well as being fed by them. The user has extensive involvement with the experimental acquisition component, and this insight is passed into the requirements review, design review, and documentation processes of the serial program. Therefore, there is feedback through both document preparation and hands on activity. There are several requirements validations which can support an almost constant process of requirements validation, and the acquisition strategy of a standard or serial acquisition can remain in place. We believe the opportunistic acquisition strategy has the correct approach to the application and potential high-payoff for C^3I systems. It is basically a prototyping life cycle which allows well definition of the government/contractor business relationship.

Planning

Planning and management techniques tend to be ineffective as currently applied to C^3I systems. There is a limit to how finely we can decompose the goals of the C^3I program. We have no track record, no metamodel, of the problem domain. The results of requirements analysis remain unclear. As such, a priori planning at a very decomposed level doesn't make sense. We must use management techniques much more akin to real-time command and control, which tend to emphasize reactive strategies more than the approaches currently in vogue. In order to support a reactive management style, risk management and prediction becomes important, and we must withhold a detailed a priori commitment of resources so as to be able to meet risks as they become realized. Therefore, the ability to integrate schedule and cost estimation tools together with technical performance management tools is crucial. The program manager must be given the tools necessary to evaluate immediately where he is technically and drive out new schedules, new cost estimates and technical approaches which are appropriate. This runs almost completely counter to what is taught in management schools and what appears to have been the foundation of management philosophy for past larger aerospace programs.

Testability

The notion that we can design a system and then test it for the complex distributed systems of today is inapplicable. The design of a complex system capitalizing upon the distributed architectures that allow inclusion of special purpose processors may create an overly complex architecture. Such a system can have hundreds of thousands of accessible states. Perhaps as many as fifty or sixty thousand are high probability states. With the advent of fault tolerance as a design imperative, we are actually creating additional states of the hardware and software system which may be accessed. Therefore, we can only expect, especially with the increase in specialized small computers bound through LANs which have fault tolerant aspects, a proliferation of the number of accessible states. Current test philosophy is based on a notion that one may enumerate the accessible states of a system and create test procedures driving that system into the high probability accessible states. At that point, the performance of the system as well as whether or not it can be partitioned into required states can then be assessed. This doesn't make sense for the systems of today. We cannot enumerate all the accessible states; we cannot identify the high probability accessible states; we do not have the time nor the appropriate drivers to place that system into all the states which we have identified. Therefore, we are forced to consider "testability" as a requirements criteria and a design criteria. We must choose to design testable systems and we must choose to specify testable systems. But determination of design attributes resulting in testability are not the subject of this study; similarly, neither is determination of the qualities of a specification which lead to a constraining of the design solution towards testability,the subject of this study.

Estimation

A further problem of C^3I systems is that current software estimation techniques cannot always be applied so as to yield usable results. It is well known that software estimation techniques tend to rely on line of code estimates. A line of code estimate has some relationship to the underlying complexity of the problem when the problem domain is well understood and we have developed significant similar programs. Real cost to a program is founded in the complexity and scope of the requirements. Current software estimation techniques do not rely on some structural model and/or quantification attribute of the requirements. Perhaps at some point in the future they will and the function point method of estimation has promise. A further problem for C^3I systems is that we do not understand the requirements domain even as well as we do the weapon system domain. Within the weapon system domain we have notorious failures of the lines of code estimates. Therefore, applying current software estimation techniques to C^3I systems will in all likelihood fail.

3.7 Conclusions

Our examination of the basis for error and problems on C^3I system development has shown that requirements definition for those aspects of the system directly supporting the site user's cognitive processes is key. Any process which serves to mitigate problems in the communications process between user-developer, acquirer-user, and acquirer-developer would be a major help.

4.0 RAPID PROTOTYPING AS A SOLUTION

4.1 Solution Set

Given these general problem areas and types, what are the potential solutions and how does prototyping fit within the solution set? Basically, as shown in Figure 4.1-1, the kernel activity leading to better systems is requirements validation. We have examined what this entails in order to better define how to use prototyping to support C^3I system development.

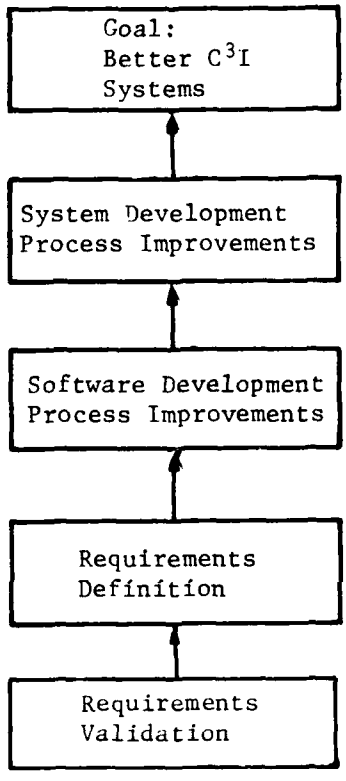


Figure 4.1-1
The kernal of the problem of developing better C^3I systems is requirements definition.

Problems in software development do not arise in software. The key driver of software problems in development is the destabilizing effect of higher level requirements which are themselves unstable. The requirements validation process has three components, as shown in Figure 4.1-2. These parts are requirements identification, requirements representation/expression, and requirements evaluation. Much work has been performed recently in the area of requirements expression. Use of structural techniques is mature. The role of modeling and the use of test beds to support requirements evaluation is mature in several areas, though its use is not widespread. It is in requirements identification that the majority of work remains to be done. Few tools exist to aid a user/developer in deciding what is needed. Our analysis has shown that there are three attributes or facets of the requirements identification problem.

The three attributes of the requirements identification problem are Consistency (Will the system hang together? In other words, do the requirements mutually agree, or do they contradict one another.); Completeness (Do the requirements adequately define what is to be done? Have we defined a C^3I system or left anything out?); and Validity (Will the system satisfy its intended users, or are we building the right C^3I system?).

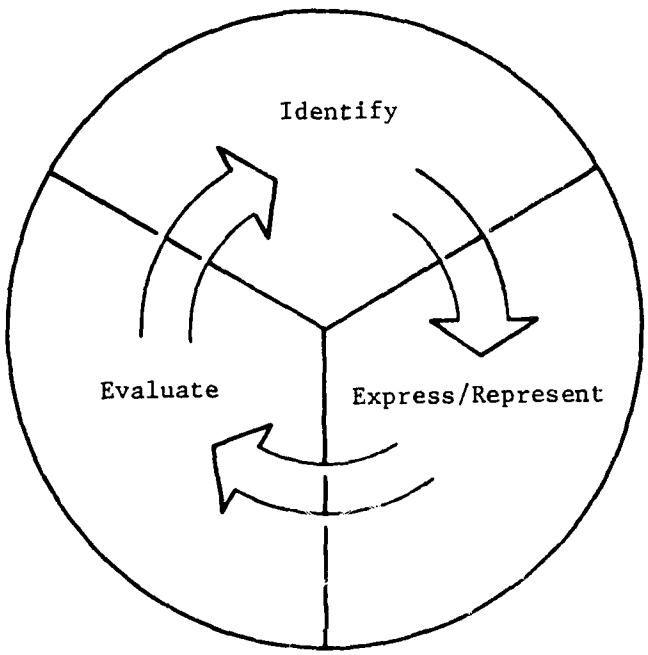


Figure 4.1-2
The Three Aspects of Requirements Validation

These issues might be expressed as: Are we building a system, are we building a C^3I system, and are we building the appropriate C^3I system.

The ways to answer these questions and the tools to support the process are lacking. Rapid prototyping offers a means to improve the process of answering the questions by allowing construction of large portions of a system's functionality early and cost effectively. The system so approximated can be the subject of experimentation to determine its attributes. How to choose the approximation is key to the character of the prototyping activity. There is no clear consensus of how this choice should be made, and it remains in the realm of art. Software exists in several forms/dimensions:

- performance
- resource requirements
- functional description
- logical description
- algorithms
- actual code

Each aspect may be represented to the detriment of the ability to represent or evaluate the others. Given constant resources, the better prototyping approach would represent, exercise, or create software prototypes exercisable in all of these aspects. Figure 4.1-3 shows a cartoon of how prototyping is to do its job. The artfulness in the process is associated with what is included and left out in the skeleton functionality.

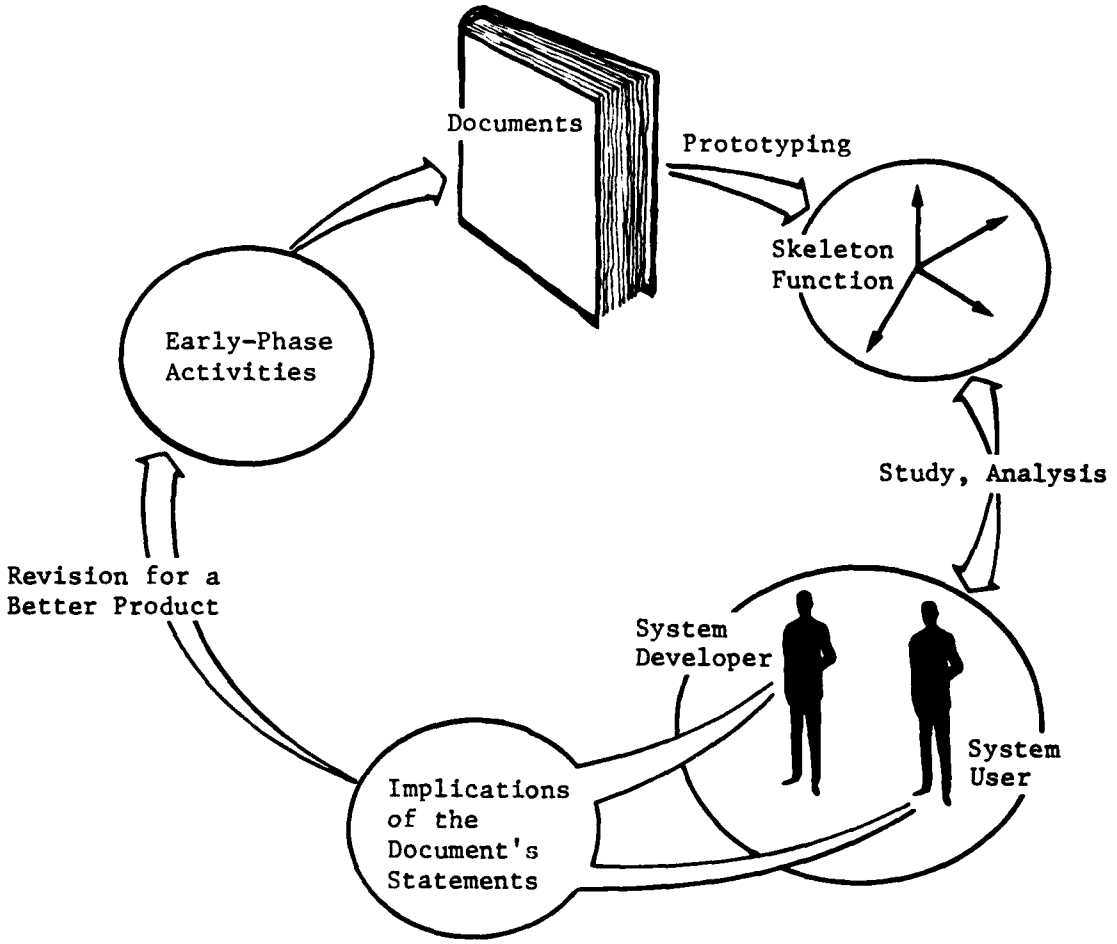


Figure 4.1-3
Prototyping is a forward-looking activity within the development process.

Alternative Solutions

We perceive three alternatives to solving the problems of C^3I system development. They are not exclusive but represent general partitions of the solution set:

- extending and deepening the requirements analysis process;
- vesting the using command with acquisition authority;
- automating the requirements process through, say, knowledge based techniques.

The first alternative involves increasing the time allocated to requirements definition as a development phase as well as increasing the effort/cost of such a phase. During this period, more and more extensive activities of the type currently occurring during requirement analysis would take place. Various degrees of automated support of a record keeping nature may occur. This would involve such aspects as more use of project data base, electronic media, and automated consistency checking.

There is evidence to suggest that the use of Ada as a target language will extend the requirements process. This planned increase should occur if Ada or a subset is used as a PDL. During this extended period there would presumably be additional contact with users and more opportunity to consider alternative solutions. The likely effect is to increase the program cost at the early phases in order to minimize the high cost of integration and test as well as reducing risk. Most proponents claim that this approach may reduce total program cost over a complete system life cycle.

Use of evolutionary acquisition is a subcategory of this theme of increased requirements definition effort. This process, as shown in Figure 4.1-4, consists of a sequence of carefully phased blocks. Its intention is to limit development risk and to virtually increase the requirements analysis period. The approach has had large success on OASIS recently.

In the second approach to reducing development problems, the using authority would also be the acquiring authority. This in principle reduces the number of intermediate translations of the requirements, reduces the communications problems, speeds up the turnaround time of the feedback loop, and costs less (by reducing the involvement of other organizations). The trade- off and source of risk is the reduced technical sophistication of the user and his reduced experience in program management. The recent TAC experience in fielding CAFMS is a notable data point. CAFMS was an in-service acquisition fielded in considerably less time than the average 10 years for a C^3I system deployment. The cost was also considerably less, even discounting some cost and time reduction due to the heritage of TACC/AUTO used by CAFMS.

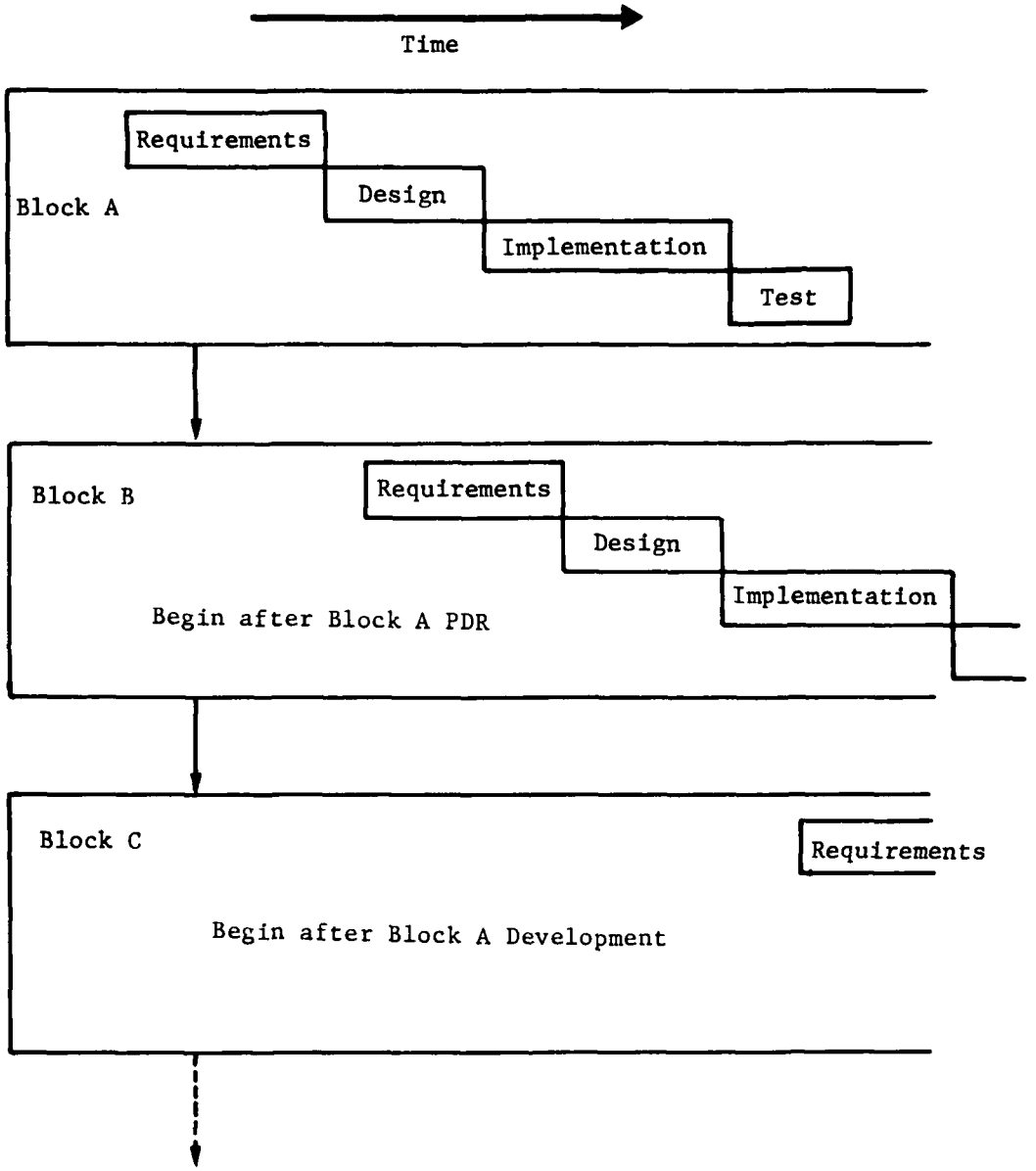


Figure 4.1-4 Evolutionary system acquisition is basic to C^3I systems.

The constant watch program for PACAF and the EIFEL I program for USAFE involve significant direct user control. While not a C^3I system but rather a C^3I simulation system, the Warrior Preparation Center (WPC), a joint USAFE and Army program, is also a data point. The WPC is a larger effort than CAFMS and is early in its life cycle, so an evaluation is not possible. Taken together with the other efforts, however, it points to a trend toward a merging of the using and acquiring command authorities.

The third alternative is the automation of the requirements analysis process through the use of techniques such as knowledge bases. Research toward such a goal has been conducted by the Knowledge Based Software Assistant (KBSA) effort funded by RADC through the University of Dayton. The KBSA would require construction of various expert systems and meta-models, the most notable of which would be a meta-model of the structure of the application domain - C^3I systems. An intermediate stage for such an approach would use mixed- initiative knowledge based systems providing some user interaction. The time necessary to develop such an approach is projected at 10-15 years by the above group. This is certainly far from maturity.

The most likely course for C^3I system development over the next few years is a combination of all four solution approaches. This would see evolutionary acquisitions characterized by increased and deepened requirements definitions using rapid prototyping and ever more increasingly automated support environments. Finally, increasing sophistication and involvement of the using authority in the acquisition can be expected.

4.2 Definition of Rapid Prototyping

Stating a definition for rapid prototyping is not an easy task. A special edition (12/82) of the SIGSOFT Software Engineering Notes devoted to rapid prototyping did not present a consensus. Most of the articles offered their own definitions. Prototyping of hardware systems is well understood and refers to construction of a scale model of the system. Prototyping of software is different due to its abstract nature until it becomes code. Prototyping of software in the stages before it assumes this concrete shape has the highest payoff. This is due to the fact that the problems of software are in its definition, not in the production of code. Rapid prototyping of hardware is clearly the quick/early production of a scale model. Rapid prototyping of software refers to the quick and cheap production of an approximation of the software in some stage of its development.

Prototyping is a process. We believe that the term rapid prototyping should be seen as a code word for modern tool-aided software development practices which stabilize requirements and design quickly and completely. This is counter to much of the work reported in the literature which takes the view that software prototypes should be like hardware prototypes and be capable of evolution to the final product.

Rapid prototyping for systems as complex as C^3I systems can only take place within a special facility, laboratory, or test bed geared to the special needs of C^3I system definition. The tools and support environment of a laboratory would make it possible for less experienced or sophisticated people to define and build software. Similarly, it would allow experienced and sophisticated personnel to build more complex systems. This is the general

lesson of programming environments and tools to support system development. Without a specialized facility, rapid prototyping could still take place; however one would need a permanent team of productive, creative software engineers to quickly build their notion of a system on a custom basis. Current trends towards system complexity and difficulties in retaining creative personnel militate against the second solution.

4.3 Rapid Prototyping's Role in C^3I System Development

The basic principle supporting the application of rapid prototyping is that the earlier in the system life cycle problems are identified, the less costly will be the solution. Prototyping is one major way effective preplanning can be conducted. Our experience has shown the following:

1) Prototyping must be invoked at the earliest possible stage of system design and development.

2) Several cooperative (but distinct) types of prototyping are required.

3) An effective procedure must be invoked to incorporate prototyping results into the planning, design, and development processes.

Careful formulation of requirements, specifications, and design are important. But behavioral feedback from the intended user reveals information that is difficult to discover by analysis of a static system description such as a specification.

Prototyping provides a new, richer basis for discussion and statement of intent that complements normal specifications. In short, the study of the potential behavior of a system has a greater payoff in requirements and design stability than study of static descriptive information. In the traditional life cycle model, this behavioral feedback from the intended user becomes available only at the end of a lengthy development, as shown in Figure 4.3-1.

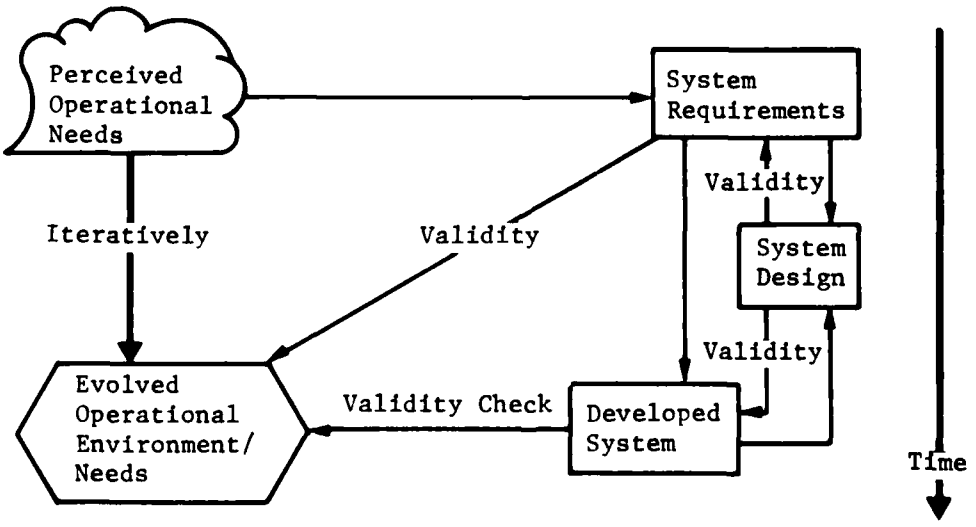


Figure 4.3-1 Validation of Requirements

Prototyping can shorten the feedback loop in key risk areas before large investments have been made in development. Prototyping can aid in these areas: Definition of system requirements; establishment of criteria to evaluate system performance for a given system concept; simulation of operational concepts; operator-to-system interaction; system architecture and operation procedures; capability to demonstrate system operation to provide a common base of dialogue between designer, developer, and decision maker.

Just as the problems provided by C^3I system development are varied, the tool set and the approaches to creating an executable prototype are also rich.

Prototyping Drivers - Schedule maintenance, technical compliance, operational success, and maintenance of cost and performance milestones are drivers of prototyping activities. The results of prototyping activities should aid in identifying and validating requirements soon enough to maintain program schedule. In an evolutionary acquisition, this means that the timestepping will force prototyping to start earlier in real time, which, in turn, is the driving requirement for earlier existence of the prototyping tools. This also points towards the need for a laboratory.

There is a cycle of requirements identification, expression, and validation that returns to identification as was shown in Figure 4.1.2. Prototyping can assist in each part of the process. A hierarchical description of C^3I functions supplemented by a help file aids in requirement identification. A structured means of requirements statement, such as that available in SREM and Designer's Workbench, aids in requirements expression. Tools such as system performance models, structured problem statements based on state-transition diagrams or Denver Aerospace's C^3 Systems Laboratory aid in requirements validation. Certain high-level procedural languages, if they are mature, such as Prolog (for data base and query systems) provide the capability to express requirements in an executable form.

5.0 APPROACHES TO PROTOTYPING

5.1 Types of Rapid Prototyping

We studied all four present approaches to prototyping. An analysis of each follows:

Very High-Level Languages (VHLL) - The first is through the use of special-purpose high-level languages. These languages are either procedural in nature, like PASCAL or Ada, or are data-type oriented like the graphics tools GRASYS and XPL/G and the specification tool GIST. The ways these tools are used to prototype are as follows:

1) Identify requirements (not part of the tool);

2) Express the requirements in the language;

3) Execute the resulting program.

Fundamentally, the requirements specification is a program that is executable, and therefore, a prototype. Use of these tools for C^3I system applications is not yet mature. Use of the tools for any specific application, e.g., message processing, is similarly not mature. Effective use of such tools requires a robust application specific support environment to aid in identification of requirements. Such tools linked to the existing high-level languages do not exist. Further, these tools have only been proved in extremely limited applications. Our work has categorized this high-level executable specification approach as immature. There is one exception. The language Prolog has been developed to support data base system development. Data base management systems and general command language access mechanisms can be constructed very easily in Prolog.

A Prolog-expressed set of requirements is immediately executable with good performance. Prototyping of data base system components is a key to intelligence system applications and is especially important in analyst support environment applications.

Modular Algorithm Libraries - The second approach to prototyping is through the use of modular algorithm libraries. This approach is highly site-specific, depending on the algorithm heritage available. These systems could be used as follows:

1) Identify requirements (not part of the tool);

2) Express requirements in an informal document;

3) Browse the library to select heritage;

4) Assemble a software package.

Such approaches require a fairly longstanding commitment to maintaining a code heritage and support library. The function of the library is to maintain documentation and aids to control the software interfacing between the modules. Our study has not shown any major libraries that support such an approach for any problem or generalized problem area.

Data-Base-Driven Tools - A third approach is to use data-base-driven hardware and software tools that, when instantiated, provide varying degrees of functionality. The area where these tools have success is in characterizing the man-machine interface. For the man-machine interface (MMI) application, a computer and display-device hardware suite host a picture generator. The tools are used as follows:

1) Identify requirements (not part of the tool);

2) Express requirements in an informal form;

3) Initialize a scenario in the test bed;

4) Refine the scenario "hands-on."

The test beds have most widely been used in the MMI areas. Examples are TRW Corporation's FLAIR and Martin Marietta Denver Aerospace's C^3 Systems Laboratory. The world of MMI test beds has two components. The first uses static display frames and static graphic entities to create a sequence of fixed images that can be played back to give an illusion of movement. Online modification of scenario dynamics is not possible because the dynamics are provided through "flashcarding" the frames. TRW is currently extending the picture building portion of FLAIR to include dynamic models of elements such as ships. The second approach uses an object-oriented graphics builder and an extremely elaborate data base to hold definition of pictures and overlays. Either (1) a vector calligraphic-quality display device such as an Evans and Sutherland, or (2) a bit-mapped device such as RAMTEK with special software is necessary.

This second MMI test bed provides very dynamic "on-the-fly" picture manipulation. Online scenario modification is possible. Denver Aerospace uses this approach in its C^3SL. The data-driven test bed approach is mature and has been proved in application to intelligence systems. Drawbacks are that there is no way the system will of itself identify requirements or the system's cost, and there is little opportunity to directly migrate solutions into development. Some identification of necessary workstation and display components occurs by virtue of the tool's underlying meta-model of a class of C^3I displays. This provides a kind of completeness information through offering subelements of an abstract workstation (maps, overlays, alarms, menus, etc.). Significant investment must be made in either approach, and the dynamic, online modifiable test bed is more costly. Clever implementations could be made using modern technology and the heritage of our C^3SL to provide the same functionality for less cost than C^3SL's development cost.

Structured Problem Statement/Analysis Tools - The fourth major approach to prototyping uses structured problem statement/analysis tools. These methods characterize the system to be prototyped in terms of a graph, procedure, state-transition diagram, or hierarchically structured data base. Often, a large prototype constructor may interleave these techniques. They are all equivalent to describing the problem to be prototyped as a finite-automata problem. Certain extremely general discrete-event-based model-building tools fall within this approach. These have been referred to

in the literature as application specific simulation languages. SAINT is one example. The tools are used to:

1) Identify requirements (some help provided by the tool);

2) Express a problem or requirement in the tool;

3) Run prototypes.

At times, a structured requirements expression language may be linked with the tools. This is the case of the Software Requirements Engineering Methodology (SREM) approach. We evaluated SREM under contract to RADC to determine its applicability to Air Force C^3I system developments. We determined that generation of the "beta" system descriptions was insufficiently automated, and the tools (model builders) to construct the betas were not general enough.

The structured problem statement/analysis approach is mature and is proved by application to C^3I systems. Our investigation showed that it was the most cost-effective for application to C^3I problems.

5.2 Taxonomy of Approaches

The previous section provided a synopsis of the main portions of prototyping approaches. We have prepared a detailed taxonomy of the world of prototyping as shown in Table 5.2-1. Its purpose is to provide a means to classify prototyping approaches in a fundamental manner and to supplement the more empirical discussion of the previous section. There are six dimensions or aspects to the field of prototyping.

Table 5.2-1 High-Level Prototyping Taxonomy

Level 0 Prototyping =
(Evolution Dimension + Functionality Dimension + What Entities Addressed + Automated Prototyping Dimension + Which Phase + Parent Specification)

Level 1 Evolution =
(Refinable to The Product, Discardable)

Level 1 Parent Specification =
(Prototypical Specification, Normal Specification)

Level 1 Functionality Dimension =
(Non-restricted, Restricted)

Level 1 How Defined =
(Separate Requirements Generation, No Separate Requirements Generation)

Level 1 Which Phase =
(Statement of Need, System Definition, Software Requirements, Software Design, Other)

Level 1 Which Entity =
(Operational + Functional + Logical + Algorithmic + Physical)

Table 5.2-1 (concl)

Level 2 Refinable To The Product =
(Language Evolution + Host Evolution)

Level 2 Discardable =
(Effective Procedure To Obtain Output, No Such Effective Procedure)

Level 2 Prototypical Specification =
(SON*, System Spec*, Interface Control Document (ICD)*, Ops Concept*, B1*, B5*, C1*, C5*, Other)

Level 2 Normal Specification =
(SON, System Spec, ICD, Ops Concept, B1, B5, C1, C5)

Level 2 Separate Requirements Generation =
(Effective Procedure To Obtain Prototype, No Effective Procedure)

Level 2 No Separate Requirements Generation =
(Templating, Executable Specifications, Translation)

Level 2 Restricted =
(Restricted Functional Elaboration Scope Unrestricted, Complete Functional Elaboration Scope Restricted, Restricted Functional Elaboration Scope Restricted)

Level 3 Templating =
(Hierarchical, Non-Hierarchical)

Level 3 Language Evolution =
(Language Change, No Language Change)

Level 3 Host Evolution =
(Host Change, No Host Change)

Level 3 Executable Specifications
(Mathematical, Operational, Axiomatic)

Evolution Dimension - A prototype is refinable to the eventual product, or it is discardable. If it is discardable then there is an effective procedure to obtain the results or insights from its use, or there is not. An example of an approach where there is no such effective procedure would be an MMI prototype developed on a test bed very different from the eventual target ADP environment. It may provide general information on the type of system desired, e.g., a raster display vs. a vector display, but the tool itself does not print out the words "raster display recommended." This sort of approach provides its information through users reacting to it and communicating their impressions in an unstructured or free-form manner. An example of an approach including an effective procedure would be the same test bed with a file in which to collect user impressions, or a performance prototype which collects data translatable into the resource requirements to run the software represented.

If the prototype is refinable to the product, then there is a possible sequence of programming language changes to evolve the prototype to the product, and/or there is a possible change of host processor/ADP to the target system. This is the case of the Naval Research Laboratory's (NRL) Secure Message Processing system prototyped in FRANZLISP. Of course it is possible that the original prototype is a major portion of the desired target system requiring neither a language change nor an environment change.

Parent Specification - We can classify a prototyping approach by what sort of input it requires in terms of a specification. This refers to the approximate level of detail in terms of normal life cycle documents necessary to use the approach. This parent specification is either a working specification/document (appearing in the taxonomy with asterisks under Level 2 prototypical specification), or it is a normal document. A key point is that working documents can be available earlier in the life cycle than normal documents. Additional differences involve rigor and format. One can see the difference in comparing a working document containing some B5 together with A and B1 level data with a formal B5 document.

Phase - This dimension refers to the earliest or best phase of the development cycle which the prototyping approach supports. The phases are listed as statement of need (referring to early formative activities), system definition, software requirements, software design, or other. This last refers to later life cycle elements. An example of prototyping in the statement of need phase might be supporting constraints analyses on the time to process new reports by a tracker-correlator in a proposed correlation/fusion center.

Which Entity - This aspect refers to what level of abstraction of the software is being represented or manipulated. The subcategories are operational, functional, logical, algorithmic, or physical (code) as shown in Table 5.2-2. The earlier a software system can be represented and evaluated, the more stable subsequent detailing will be. Most prototyping tools/approaches currently discussed in the literature deal with representing software in the algorithmic stage of its existence.

Functionality Dimension - This aspect of an approach should be fairly intuitive as it refers to the portion of the overall system that is being prototyped. The represented functionality is either restricted to a subset of the total system or not. In general, prototyping of the total system functionality as through a hardware mock-up is very unusual in the C^3I world. It is not at all unusual in the world of aircraft, simulators, cars, or missiles. Non-restricted prototyping is equivalent to the "build it twice" philosophy of system development.

Table 5.2-2
Software takes shape through several levels of abstractions.

Subcategory	Entities Described	Form
Operational	Operability Requirements	Document
Functional	Functional Requirements	Document
Logical	Data Flows	Graphic
Algorithmic	PDL	Document
Physical	Code	Tape

If the prototyping is restricted, there are three states of this restriction depending on whether the scope or functionality of a system component is being represented. Scope restricted refers to limiting the number of functions represented, e.g., representing the message handling aspects of a C^3I system but not the track generation function of the system. Restricted functional elaboration refers to the extent to which the represented function is exercisable; or, similarly, the extent to which it is approximated. Most prototyping currently falls into the restricted functional elaboration/scope restricted category.

Automated Prototyping Dimension

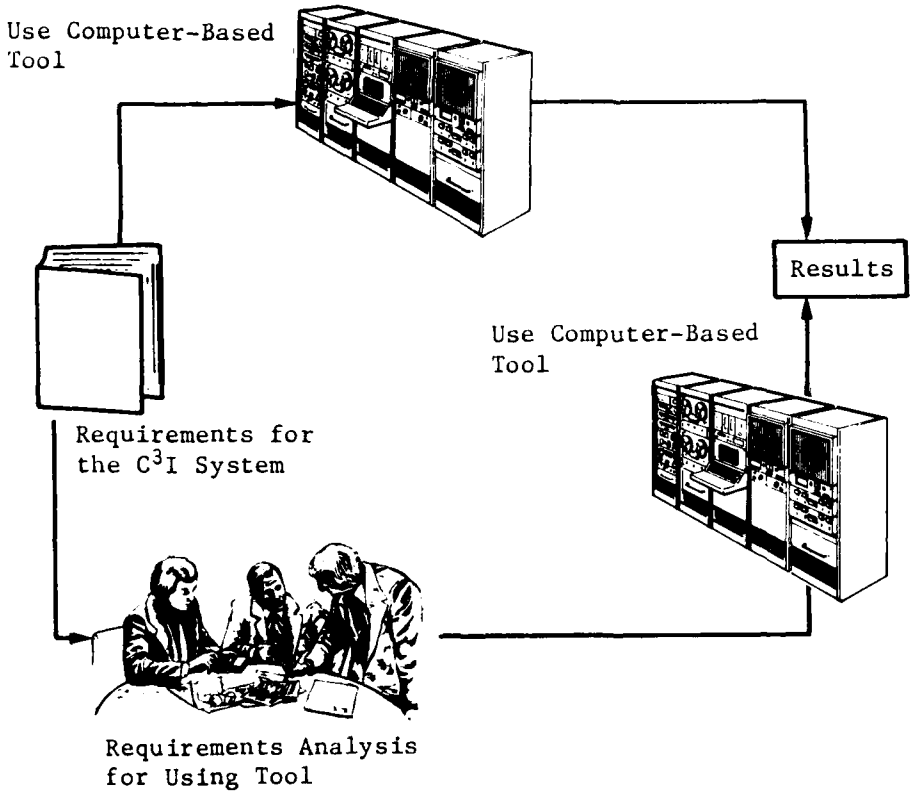


Figure 5.2-1
Automated prototyping can occur only where a computer tool can directly process the system's requirements.

Figure 5.2-1 shows the two parts of this dimension. Details of approaches to automated prototyping have received significant attention in the literature. The level one decomposition considers whether or not, given a parent specification, the prototyping tool can directly process it. If not, then there must be a new mini-requirements analysis process to define how the tool will be used. Until systems such as Dr. Balzer's GIST concept become mature and are appropriately instantiated for C^3I systems, separate requirements generation will be the normal course of prototyping. We discuss this distinction more fully in a discussion of our preliminary methodology later in this report.

If no separate requirements generation is required, then the prototyping tool can be instantiated one of three ways: Templating, executable specifications, or translation. Translation refers to the existence of a front-end to the prototyping tool which can parse a specification and translate it into an executable form automatically. This might be a natural language processor. Templating refers to a human augmented, totally guided procedure to instantiate the prototyping tool (from a parent specification or quasi-specification). This templating is either hierarchical or not. An excellent candidate for templating based prototyping tool instantiation is the MMI. A great deal of analysis has been performed on the structure of determining MMI requirements against the possible universe of solutions. The Mitre work of Sidney Smith is foremost in the area.

Many workers in the field evidently believe the use of executable specifications is the only way to perform rapid prototyping. The project documents would be written in a format or language which was a high level language. They would be human readable (after appropriate training) and executable (code equivalent). We are far from realizing the potential of this approach for C^3I systems. Once an appropriate language has been selected, a support environment and/or set of C^3I system specific constructs must be defined and built. There are four bases for executable specifications. They correspond to the systems of defining the semantics of the languages. These categories are mathematical semantics (also known as denotational), operational semantics, the axiomatic approach (implicit semantics), and informal (implied semantics).

5.3 Analysis Criteria

In order to evaluate the different approaches and instances of prototyping, we developed some criteria as shown in Figure 5.3-1. For each criterion the potential valuation is also shown. Note that there are three subfactors associated with relevancy to requirements evaluation. We noted what life cycle phase the approach supported, what aspect of requirements analysis it supported, its maturity, and its immediate applicability to C^3I systems. Valuation judgments were made by reviewing critical literature and studies, tool and approach documentation, and interviewing task leader/program manager/ IR&D principal investigator personnel at Denver Aerospace. In the course of our literature search to support the program, we have collected and reviewed over 350 documents.

Two approaches were not ranked. The first is the use of software remaining from previous developments (heritage software) or algorithm libraries. This is site specific and no evidence occurred in the literature

where a company had made use of such an approach. The second approach was reduced functionality quick build. This approach refers to creative and productive teams of software engineers who, using a build-it-twice philosophy, hypothesize a portion of a system's software. No doubt this occurs on a relatively widespread basis but is insufficiently disciplined/structured to value. An example might be NRL's Secure Message Processing System work.

- Applicability to C^3I (0,1)
- Maturity as a Discipline (0,1)
- Relevant to Requirements Identification (0,1)
- Relevant to Requirements Expression (0,1)
- Relevant to Requirements Evaluation (0,3)
  - Consistency
  - Completeness
  - User Validation
- Supports Study, Preliminary Design Phase (0,1)
- Supports Detailed Design, Test Phases (0,1)

Figure 5.3-1 Criteria for Evaluation

5.4 Evaluation of Approaches

The results of the valuation of the instances of the approaches for the criteria were collected in a table. This is shown in Figure 5.4-1. The scores were collected into a raw score which was weighted by dropping out the contribution of those scores related to support of the detailed design-test phases. This was based on a determination that high payoff for prototyping's use is early in the life cycle. We further weighted the scores by dropping out the immature technologies. The weighted scores were then combined to give an average by area for those technologies which were rated.

Formal Rapid Prototyping Languages

We have broken the area of rapid prototyping languages into three subareas: general, graphics related, and other. In the general category tools are basically built around subsets of high level languages which are procedure oriented. The approach consists of using a subset of Ada or PASCAL as a PDL. A notable example is the work by Taylor and Standish at UC Irvine. For application to C^3I systems development, however, the concept requires development of a special set of functions. This work has yet to be done. Without this work, the approach is restricted to support of detailed design and not preliminary phases. It is essentially a way to facilitate, to a limited extent, the expression of requirements.

Table 5.4-1 A Weighting of Approaches

			Applicability	Maturity	Supports ID	Supports Expression	Consistency	Completeness	User Validation	Five-PD	DD-T	Raw	Raw Weighted	Average by Area
Formal Rapid Prototyping Languages	General	Taylor's Use of Ada Subset	0	0	0	0.5	0	0	0	0	1	1.5		4.75
	Graphics Related	- Mallgren's XPLG/GRASYS	1	0	0	1	0	0	0	0.5	-	2.5		
	Data Management Related	- PROLOG	1	1	0	1	0	0	0.5	1	0	4.5	4.5	
	Other	- Balzer's GIST	0	0	0	0.5	0	0	0	0	1	1.5		
		- SREM	1	1	0.5	1	1	0	0.5	1	0.5	6.5	5.0	
Structured Problem Definitions	Hierarchical Models	C^2SAM	1	1	1	1	1	1	1	1	0	8.0	7.0	6.0
	Relational Models	FAM, AUTOIDEF	1	1	0.5	1	1	1	0.5	1	0	7.0	6.0	
	Graph Theory or State-Transition Models	GOM, GPM, PERCAM	1	1	0.5	1	1	1	0.5	1	0	7.0	6.0	
	Functional Test Bed	C^3SL, FLAIR, GIDS	1	1	1	1	0	0	1	1	0	6.0	5.0	
Heritage S/W	By Area (Application)	-	Note: This is site-specific and tied to quick-build below. We do not quantify it here.											
Scenario Generators & Test Harnesses	By Area (Application)	- Denver Aerospace's SMARTS MSG System	1	1	1	0	0	0	1	1	1	6.0	4.0	3.0
		- TYC-16	1	1	0	0	0	0	1	0	1	4.0	2.0	
Quick-Build	Reduced Functionality	Ad-Hoc Approaches	Note: This is site- and individual-specific, so it is not quantified.											
	Full Functionality	Dart System	0	1	0	0	0	0	0	0	1	2.0	1.0	1.0

The logic programming tool Prolog is popular in Europe. Our version was obtained from the University of Edinburgh. It has been shown to be valuable in expressing data base system and command language requirements in an executable form. The utility of Prolog is in the ease of using it to express first and third normal form relational descriptions of a data base system in the Horn clause subset of the first order predicate calculus. It is up to the user to provide the C^3I related constructs, however.

More work has been performed in the area of graphics specification languages which are executable. Here, the work has proceeded to the level of defining the data objects and operations to allow manipulation of picture components at the level of CORE graphics primitives. Examples are Mallgren's XPL/G work at the University of Washington and Ohlson's GRASYS at Texas A&M. There is applicability of these to C^3I systems. Maturity is lacking, however. Also, there is no facility to support requirements identification. Early phases of the life cycle can be supported through this work, but it is restricted to preliminary design.

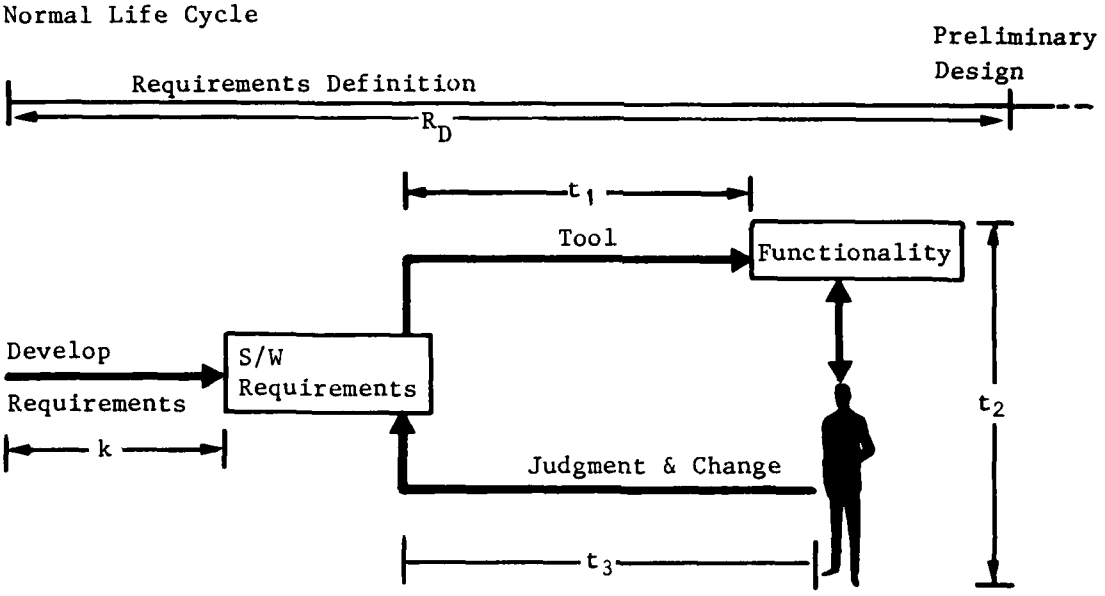
The category of "other" embraces approaches such as Balzer's GIST at USC and the TRW SREM concept. Balzer's work falls into what may be a separate subcategory of knowledge-base-supported rapid prototyping. GIST uses the operational (lambda calculus) approach to defining semantics of the language. This work is not currently applicable to C^3I systems as, again, the analysis of the application domain has not been done. It is far from maturity. It supports expression of requirements and, when complete, will support the formal validation of specifications.

The SREM approach is important and consists of a methodology and several tools. It is applicable to C^3I systems and is mature. In the sense that it enforces a structure for expressing stimulus/response network requirements (the R-NETS), it supports the identification of requirements to a limited extent. It provides consistency checks and, through development of the simulations, allows a degree of uses validation. There is no "world model" contained in the approach, so completeness checks are not possible. The data structures of the support tools make it difficult to capture concurrency and parallel operations - they enforce a sequential world view. SREM scored well, however.

Quick Build

We have already discussed why reduced functionality quick builds were not ranked. This approach is part of the build-it-twice or built-it-several times philosophy. An example of a full functionality prototype is found in the Naval Research Laboratory's (NRL) use of FRANZLISP to prototype a secure message processing system. The NRL project chose a powerful object-oriented system which hides the data manipulations to quickly construct the software. They coupled it with a specification methodology (manual) using state transition diagrams. Of more interest as a general tool is the General Dynamics DART system and its extensions at General Motors Corp. The DART's concept is to accept structured expression of presumably validated requirements. Given an implementation of a design instantiating those requirements, this code can then be refined and translated to another target machine. It is mature in retargeting code and not mature in developing code. The major drawback is in the presumption of validity of its inputs. It achieves prototyping by quickly getting code for a full system or accelerating

the later life cycle phase. It is not specific for C^3I systems. The idea behind this approach is a notion of prototyping most widely held by practitioners believing the problems of software development arise in software. They do not. Figure 5.4-2 shows the quick-build view of prototyping. If the cycle time $t_1t_2t_3$ is small, there is a chance the approach can influence the system development. This notion of small can be further quantified. In order to "zero-in" on requirements the cycle must be run through several times, each cycle identifying misperceptions in the requirements. The sum of these times, tp, must be smaller than or equal to the time allotted for requirements definition. The problem is further exacerbated by the quick-build approach's presumption of validity of the majority of the input requirements. The general area of quick-build did not score well.



$$t_{prototyping} = k + n(t_1+t_2+t_3)$$

Where n Is the Number of Iterations of Prototyping.

Note: $t_{prototyping}$ must be very small with respect to normal life-cycle activity times, or $\Sigma t_p << R_D$.

Figure 5.4-2
The time available to produce results is critical.

Scenario Generators/Test Harnesses

Within this category we find forms generation systems or forms management systems. These packages, such as DEC's FMS, are generally bundled together with DBMS or office automation packages. They allow the user to define a screen format or template which is usually restricted to instantiation with alphanumeric data. Some systems also provide a way to populate the defined forms from a file. These systems have limited utility in supporting development of the more operationally oriented C^3I systems as they lack means of defining graphical form components. For message processing or some intelligence applications, however, such forms generators have utility. Once the set of forms corresponding to the set of displays has been defined, they can be instantiated and sequences played back. This provides an effective way of communicating to the user what he will see in the eventual system. The utility of this approach has been proven on Denver Aerospace's SPADOC 4 definition phase work and the OASIS program. In both cases special graphics generators were built.

This approach is useful and it can improve the quality of user interface developments. It tends to be very cost effective and fits well within the object oriented approach to software development.

In the category of Test Harnesses there has been a lack of success at finding examples outside Denver Aerospace. The issue is not that they don't exist, but that they are not widely discussed in the literature. The basic approach makes use of extensive simulation software to create, for example, a system's message environment. This role in rapid prototyping is limited, however, because of the enormous effort required to define the data bases for a new application. The Simulation Monitoring Analysis Reduction and Test System (SMARTS) is being developed for TAC at Langley. It is a system to produce, at the packet level, message streams for a variety of TAC C^3I sites. The use of the system will be in conducting and prototyping exercise scenarios. Its strength as a prototyping tool is in user validation of requirements. Rapidity, in the sense of other tools, is not obtained. Another tool is the Denver Aerospace TYC-16 message formatting and processing system. This provides a quick formatting and processing of certain JINTACCS messages such as JANAP128. As a component of a larger testbed to prototype message processing systems, it has merit. In the same vein as these tools is the Offsite Test Facility (OTF) of NORAD for the 427M program built by Ford Aerospace. This provides a training and system test replica of the NCMC message processing system. It could be used in ways similar to Denver Aerospace's SMARTS. The general category scored well but applications are restricted due to the level of detail and difficulty of use.

Structured Problem Definition/Analysis

This area, as discussed in the overview section 5.1 above, is most promising. Section 5.5 of this report discusses some details of the approach's foundations. There are four subcategories: Hierarchical models, relational models, graph theory or state transition approaches, and functional testbeds. They can be considered crude languages whose only syntax is table or menu population. Each encompasses a model of the C^3I application area, and, in fact, the tools may be grouped on the basis of their modeled work-views. They have in common the fact that they all use an informal means

of defining or implying their semantic structure. They are all means of representing objects and operations, and all are built on top of programming languages which similarly use an informal or implied semantic structure, such as FORTRAN.

Hierarchical Models

This category contains structured models of the application domain arranged in a top down hierarchy. The C^2 System Analysis Model (C^2SAM) built by BETAC/LOGICON for TAC and RADC/CO is the single tool here. C^2SAM provides a data base of functional and operational requirements for TAC/CONUS, MAC, and USAFE/CENTCOM. One uses C^2SAM by choosing a command level, site, or function. The package shows what subfunctions are performed down to the operator activity level. It keeps track of the informational input/output requirements for the function as well as which other sites could provide the necessary inputs. These inputs are referenced as specific JINTACCS formats.

The C^2SAM is a very strong tool to guide the user in identifying requirements. Various consistency checks can be run on the defined systems. The structure of the data base guarantees completeness to the extent that the data base is complete. The data bases have been validated by the subject Air Force commands. C^2SAM does not provide facilities to generate code, and its principal use is in passing from SON to validated quasi-B5 data. Integration and development of ICD's are well-known stumbling blocks in system development, and this tool mitigates these problem areas. The C^2SAM is discussed in more detail in section 10 of this report.

Relational Models

This approach uses a very restricted underlying relational model of requirements and design entities associated with system or C^3I system development. The tools allow instantiation of project data bases which can then be used to exercise the system as a logical structure.

Boeing Computer Services under contract to SOFTECH, Inc. is developing a mechanization of the ICAM methodology called AUTO IDEF. The tool, while originally designed to support definition and study of manufacturing facilities, has utility for software development, provided that a structured technique such as the YOURDON approach is in use to define requirements or design. It permits the interactive definition or manipulation of graphical entities corresponding to project entities. The work has been extended by Pritsker and Associates for Wright-Patterson AFB as IDEF III. This effort has coupled the SLAM II simulation system to the graphical entities. This could provide a capability for production of BETA elements like SREMs if the work was modified to support software design. What is lacking is a support tool or environment for AUTOIDEF or IDEF III focusing on C^3I systems.

The Denver Aerospace Functional Allocation Model (FAM) is a tool to support requirements definition using a YOURDON technique. There is no graphics interface as in AUTOIDEF. The system allows definition of abstract software functions in terms of data flows and then exercises them in accordance with the defined concept of operations. The system was used on the SPADOC definition phase contract. It properly can influence life cycle phases from SON studies to preliminary design and can be initialized at varying

levels of detail. The strong point of the tool is its ability to support requirements expression and evaluation strongly linked to a system engineering methodology.

The category of relational models also comprises the more elaborate models of data base system requirements. The use of products like DEC's DATATRIEVE is one example. DATATRIEVE is a dictionary/directory system permitting generation of different views of a single physical store corresponding to an instantiation of a more general normal form model. An early version of data base simulation, Representation Independent Programming System (RIPS) does the same in more detail but also with more effort. It was used on the RADC/IR DIAMS-PACER contract to model and prototype a large C^3I data base. DATATRIEVE has been applied to prototyping of a joint Air Force and Navy program.

This category of relational models scored well in evaluation and is especially useful in getting to better B5 level documents.

Graph Theory/State Transition Models

The underlying model used by these tools is a state transition view. Their data bases can be populated once the states of a system have been enumerated, the transition rules established, and a graphical representation prepared. The Denver Aerospace General Operator Model (GOM) is an example. This tool represents the ways an analyst set will interact with a processor in performing their mission. It was used to support an RADC/IR contract analyzing an FTD application. The tool aids definition of a site's internal operational concept as well as performance constraints and is best applied during pre-B5 level definitional activities.

The Performance and Configuration Analysis Model (PERCAM) of TRW is another instance of this category. It is driven by event logic tree representations of systems. These are graph models representing a system's operational logic such as sequence of events, time delays, and decision nodes. This system then allows construction of exercisable simulation elements and provides insight on performance and resource requirements. As with the other instances of this class of tool, generation of design or code is not possible. Certain controlling or key algorithms may be checked out against the simulated environment, however.

The Denver Aerospace General Processor Model (GPM) represents the physical structure of a hardware/software environment. CPC or module sequences are represented and run against a model of the processor and operating system. Interleaving of operating system functions with application codes can be examined. Such things as CPU schedules, swapping, paging, and the physical and logical distribution of data can be represented. It takes a large amount of knowledge of the system or proposed system to use. We have applied it on a large classified program and on analysis of a Navy correlation/fusion problem. It is mature and supports preliminary and detailed design issues.

The category of graph theory/state transition models scores well in the evaluation but basically finds application later in the life cycle than the two previous categories.

Functional Testbeds

These systems are basically focused on the MMI/workstation elements of C^3I systems. One of the predominant problems of C^3I development is the definition of the MMI. The tools in this category allow definition of prototype MMI components which are exercisable. The purpose is to allow user validation of the artfully defined graphics displays. Application is throughout the pre-detailed design phases.

The TRW General Purpose Interactive Display System (GIDS) is a tool to use display language and display drivers to develop transportable graphics code. It allows definitions of graphical entities and scenarios. A variety of post-processors exist to calculate distances, etc., in the modeled world. This capability is applicable to C^3I systems and is mature. It supports identification of requirements and user validation.

The TRW Functional Language Articulated Interactive Resource (FLAIR) is a voice-accessed display builder. It functions by quickly building display frames which can be "played back" to provide the illusion of dynamics. It appears that FLAIR and GIDS can be used jointly to mockup MMI and scenarios. FLAIR provides the control and access mechanisms while GIDS provides the software display builders. The utility of these two tools appears mature within the early life cycle.

The Denver Aerospace C^3 Systems Laboratory is an integrated facility for definition of MMI and operational scenarios. It is controlled by an event-driven simulator. Each unique simulation uses a custom defined operational data base (ODB) that may be dynamically altered by an operator using one of the interactive function or processing algorithms within the C^3 Software System (C^3SS). The C^3SS is built around an object-oriented meta-model of C^3I system displays. As such, it provides strong requirements identification capabilities. The picture components may be controlled through a variety of access devices, but it currently does not include voice access. The system has been used on many C^3I system definitions/developments, including the SPADOC 4 Definition contract and a SPADCCS study contract.

All of these tools offer strong prototyping approaches. Consistency and completeness of requirements generally cannot be examined with them, however. Used in conjunction with other tools, this drawback may be eliminated.

As shown in the figure, the structured problem definition/analysis tools scored highest among prototyping approaches. The next section discusses how this approach relates to the current research activities in rapid prototyping.

5.5 Two Mainstreams of Prototyping Systems

Prototyping certainly supports all aspects of the life cycle, but there are varied tool requirements. Let us examine how a prototyping tool, supporting early phase activities, and its structure as a software system, needs to differ with another prototyping tool, supporting later phase activities. The technology which supports the later phases, such as passing to a preliminary design from the detailed, validated complete set of software

requirements or passing to a detailed design from a preliminary design which had been validated and is complete, or certain applications generators is relatively well understood.

5.5.1 Later Phase Prototyping Tool

Prototyping of this stage of the life cycle refers to prototyping and production of a first cut of the code. What would be necessary in such a prototyping environment would be tools to generate code from a standard PDL. Executable PDL is a good example. When this problem has been solved an additional package which allows the creation of a specification language would make sense. This could be obtained by the process of abstraction to define application specific macros for this PDL. Certainly these macros would correspond to functions which would have to occur in the ultimate application.

The emphasis in such a tool system would be on the production of a function's algorithms. Object-oriented programming systems as part of programming environments support such an emphasis. They hide the data structure underlying the objects. Data bases containing information on requirements and design documents appropriately structured and encoded in the form of one or more data dictionaries also makes sense. The ability to do traceability from these documents or their isomorphs, the structured data dictionaries, would be necessary. Algorithms that would support the construction of code from an appropriately well-defined design language or requirements language would have to be included. This points out that in order for there to be any sort of automatic generation of code, a careful analysis of the syntax and semantics of the requirements language and the design language as contained in the requirements or design documents needs to take place. Algorithms would then be sensitive to that language. Another component of a later life cycle support tool might be a library of existing algorithms or existing PDL from which one could "cut and paste" and assemble to obtain a larger functionality. For advanced systems supporting the later part of the life cycle, one could envision the inclusion of a package to do structural modeling of the developed code to support computer security analysis in accordance with government requirements. Should such a package be in place as an engineer produced the prototype code, and it passed a certain level of functional test, it would then be submitted to the security analysis package which would create a model and exercise that model and pass judgment or provide output supporting human judgment on that code fragment's attainable level of security. Another package that might be included would be a tool to analyze the produced software to determine its testability. This would have to take place against a meta model of software testability. This means that the software would be analyzed by some algorithm or algorithms which would check for certain attributes related to testability in a one- or two-pass process. The algorithms would embody known knowledge on those local constructs and more aggregate constructs which are known to cause difficulties in testability. It would be very important to integrate such a later lifecycle support tool with one or more programming environments featuring debug, edit and compile support.

Tool Structure The system structure for such a tool would consist of a parent data dictionary supervising access to subordinate data dictionaries and subordinate data bases. It further would contain an

executive which would basically manage the data dictionary. Processes could be invoked through any of the support packages to obtain the appropriate data from the data dictionary to pass into the support packages where mainly automated means would be used to construct a portion of the code, exercise it, and judge its acceptability. An additional feature would have to be the addition of some type of reconfigurable test harness that would be used to drive these pieces as they have been constructed. An open question is: To what extent would this cause a difficulty in our ability to construct a module at a time, elements of a large piece of software? It would seem that the reconfigurable test harness would have to have some way of mimicking the expected performance and expected data of the missing pieces of software.

The requirements for such a tool environment are driven by the need to have a relatively stable paradigm or meta model of the application domain. Certainly, should we possess one of these paradigms that was reasonably acceptable, it could drive the construction of support packages that would exercise or test-out the constructed pieces of software. Further, once an application is understood to a fairly deep level, it is quite feasible to construct a type of menu-based system to parametrically build it. This would allow a user to make selections from menus, fill in the blanks, much like is done in current systems to support the development of business data bases. This is often referred to as "fourth generation" or "fifth generation" languages. Note, this whole discussion presumes that we have accepted the notion that there is a need for separate prototyping tools to support different components of the life cycle. Certainly a tool that supports construction of code can be used to stabilize requirements through use of many life cycles. An argument against this as a reasonable approach to developing Air Force C^3I systems is given elsewhere in this document.

5.5.2 Early Phase Prototyping Tool

A tool which supports the early portions of the life cycle (e.g., requirements analysis or preliminary design analysis) on a system-wide basis differs from the previously discussed prototyping environment. The structure of an early life cycle support tool would be as follows: The fundamental job of such a tool would be to support the identification of requirements as well as to support the identification of design components. It would have to be a tool to support the structuring of the requirements and the decomposing of those requirements. How do we structure these early requirements? They are derived from operational requirements, mission structure and the functional requirements of the system. As we move towards a complete set of operational mission, and functional requirements we can start to drive out derived requirements for elements of the software system. Pictorial representations of requirement sets have proven to be useful in the past. Whichever way we choose to record them for this early phase tool, it should be done in a manner that can be translated to the various major pictorial representations (e.g., SADT and Yourdon). We would include hierarchical decomposition in the list. This pictorial representation is different from a structured requirements language. We should note that this tool component supports an engineer's "quick and dirty" thinking about the requirements set and is not a representation necessarily of the final set. There should be a way to proceed from this pictorial representation to data-bases containing requirements information, and published documents, or

data-bases containing structured requirements definition, or even design information.

To support the identification of interface requirements (also known as man-machine interface (MMI) requirements or operator system interface (OSI) requirements) we need another tool. This tool should allow the rapid construction of screens and control formats, yet allow the organization of these screens and control events in a very easy manner to build up complex scenarios. The analyst should then be able to play back the scenario taking a variety of paths through the screen set. This tool would support the identification of operational requirements or empirical requirements.

The next element of the tool set would be a set of tools that would allow one to encode the structure of the system and either the performance requirements or the performance expectations of the components of the system. This tool should then allow the aggregation of performance data so as to predict the attainable performance of the system or its components. Of course, this must take place with respect to a context defining scenario and there should be components within this tool set to allow the encoding of various scenarios. Some sort of scenario generation should be available for the entire early phase tool set. Scenario generation will be supported by a tool built on an application-by-application basis. For example, one would need a tool to support the generation of Space Defense Initiative related scenarios, another tool or data base to support construction of tactical scenarios, and a third to support the construction of intelligence and reconnaissance related scenarios. This tool or tool set should allow the examination of the human system component, the ADP at various levels of detail, the logical structure of the system, and the communications system supporting these elements. It should predict, using statistics, the resource requirements, throughput, queueing, and time to perform various functions as defined by the analyst.

5.6 Conclusion

Our examination of the approaches to prototyping and the instances of prototyping tools has resulted in selection of the category of Structured Problem Definition/Analysis (SPDA) tools as the most promising for application to C^3I systems. These types of tools and this approach do not appear to be areas of research in the literature. Most efforts are focused on development of executable specification languages. The work on these language systems is

intimately tied to work in formal semantics definition. This is related to the problem of proving correctness of programs written in the new language.

Figure 5.6-1 shows the development of a prototyping system as a sequence of translations. An analysis of the world of C^3I systems must be made to identify the objects, operations, and control structures necessary to best describe that world. This set is then captured in a new programming language such that its code is reasonably human readable. This allows it to function as a specification of the system in a normal sense. The ability to execute the specification on a machine can occur thanks to a compiler which maps the language constructs to register manipulations.

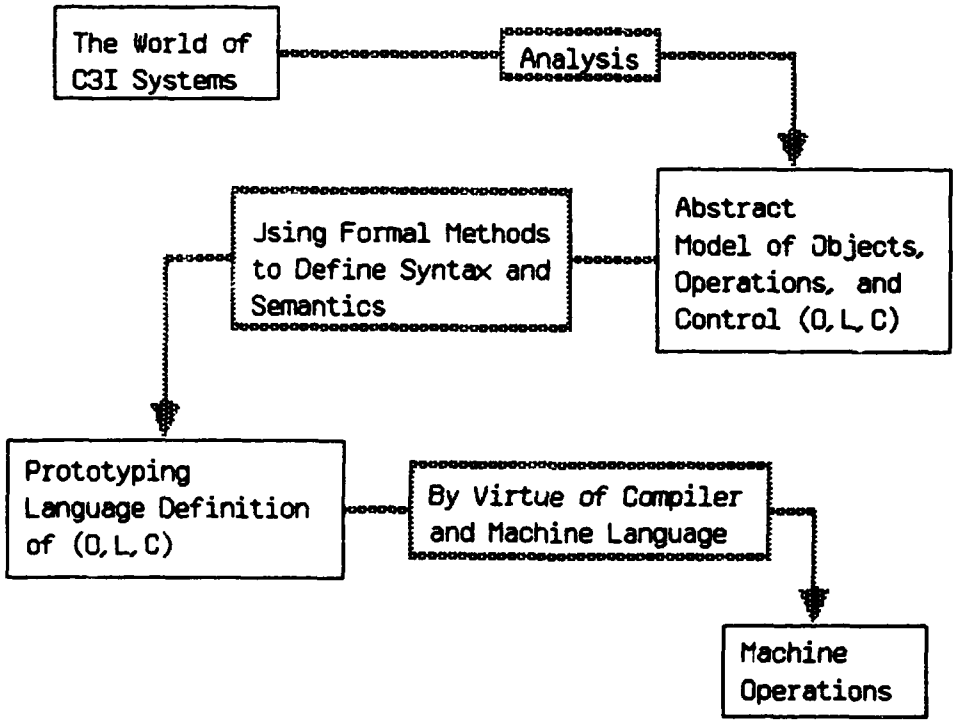


Figure 5.6-2 A view of prototyping language development as translation.

There are two problems with this new language approach. The first is that the life cycle of a new language is large - on the order of 10 years. But, more importantly, the language cannot be "best" or "natural" for C^3I systems without an in-depth analysis and abstraction of the objects, operations, and control structures of the world of C^3I systems. This is non-trivial, first, because of the complexity of C^3I systems, and, second, because the difficult structures to represent are those associated with human cognitive processes - an open question in psychology. This field does not possess models of human cognitive processes; further, most research work in the field has been in the analysis of children's cognitive processes. This problem of extracting extremely subjective information from diverse individuals and groups of people and distilling abstract models is equivalent

to the knowledge engineering problem of artificial intelligence. Predictions have ranged from 5 to 10 years for solution of this class of problems within the field. Therefore, we must consider the time to develop an adequate specification language approach to prototyping of C^3I systems to be anywhere from 10-20 years away. Further problems are caused by the course of research in this area today. All approaches to development of such a programming language focus on a general use set of concepts. When developed and mature, they will define and provide a set of objects, operations, and control structures to which the C^3I systems' objects, operations, and control must be mapped. This is the same problem faced today in using high level languages. One is constrained within, for example, FORTRAN's data-type structure and control structure. The process of design is the process of describing the application's objects, operations, and control in the limited set provided. One of the advances offered by PASCAL or ADA is the ease of defining data structures and the wide variety of data-types. In summary concerning this approach, we have to report that there are many problems yet to be overcome and few of them are currently being worked.

In comparison, consider the sequence of translations/mappings shown in Figure 5.6-2. Again, there is a need to analyze the world of C^3I systems to abstract a set of objects, operations, and control. This is translated into an application's world view through instantiation of the data base and perhaps changing parameters in the algorithms. This application is written using the objects, operations, and control structure of a modeling language such as SIMSCRIPT, SIMULA, or SLAM. These, in turn, are built on top of base language constructs such as FORTRAN (in the case of SIMSCRIPT IIF and SLAM), and then mapped to a machine's register operations. The number of translations of the abstracted objects, operations, and control is four compared to two in the previous approach, or three compared to one if we disregard the machine dependent translation.

There are some advantages to this approach, however. First, these SPDA tools exist and have undergone a degree of evolution over the last five years. Secondly, they contain a model or world view of C^3I systems which, though sometimes based on heuristics, has proven valuable within a structured system engineering methodology on real C^3I developments. This skirts the problem of abstract analysis which is the fundamental base of defining new prototyping languages.

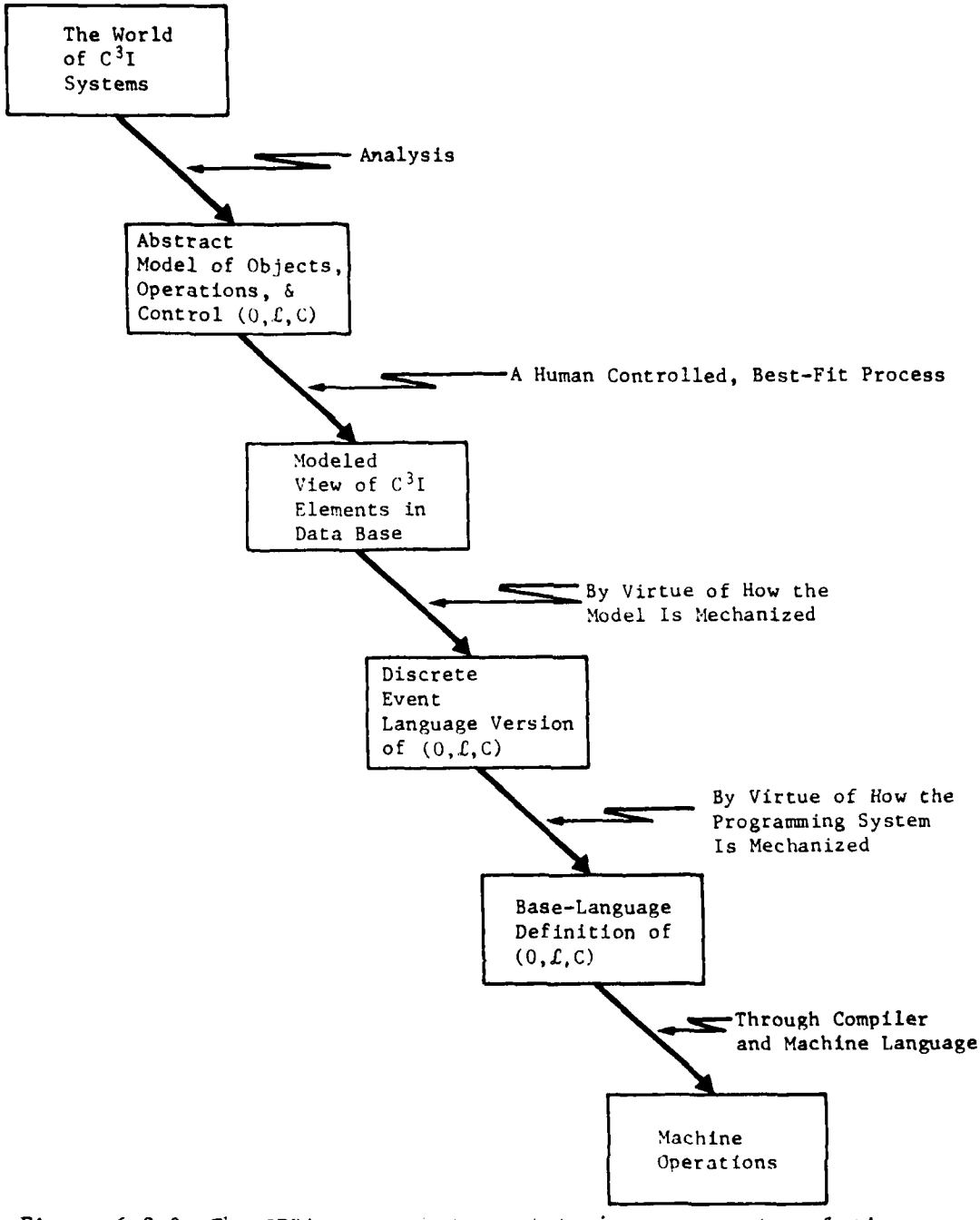


Figure 6.2-2 *The SPDA approach to prototyping seen as translation.*

By using intermediate steps, the SPDA approach approximates the translation of objects, operations, and control to an executable form which could take place if a prototyping language were available. We consider use of SPDA to be a prototype of the language approach. By putting in place a laboratory based on the SPDA approach, we would obtain the following benefits:

- o immediate prototyping support of C^3I system developments;
- o a prototype of a prototyping language system;
- o a basis for evolving and evaluating the abstraction of C^3I objects, operations, and control structures;
- o significant cost saving;
- o testbedding of the prototyping process/methodology.

In the future it may be possible to "tune" the translation steps or to sequentially collapse the translation steps onto one another. This would be a way to evolve a version of the executable specification approach.

6.0 MATCHING PROTOTYPING APPROACHES TO C^3I FUNCTIONS

6.1 Overview

In order to ascertain where rapid prototyping would have the most impact in support of C^3I system developments, we have analyzed the structure of C^3I systems. Our plan was to decompose the components of typical C^3I systems, establish which were most important, and then map these functions to the prototyping approaches/tools. We encountered some problems in this task. The notion of "importance" of a function was difficult to establish. Secondly, C^3I systems elements depend on one another to a large extent. Isolation of functions was difficult. Thirdly, some of the different functions were unable to be measured by a common standard. The qualities, attributes, and sources of difficulty were alien to one another. We were able to achieve a weighting, however, through appeal to the government's recent study activities.

6.2 Functional Decomposition of C^3I Systems

Figure 6.2-1 shows those embedded computer functions which are considered to be the kernel, i.e., common to all C^3I systems. This flow represents the bidirectional transformation of event-related data to output information. The three activities of data management, data presentation, and intelligence processing/analysis are really a cycle which may be traversed many times. This breakout of functions is common and well supported by the literature, practitioners in the field, and site users.

Figure 6.2-2 shows that these functions are made possible by other functions, some of which are allocated to humans. This figure attempts to communicate that the structure of C^3I systems functions is dense and highly dependent. Each plane in the figure corresponds to a class of functions. Certainly the kernal functions cannot take place without the underlying system management and support functions. Figure 6.2-3 shows that the kernel functions support the mission areas of the particular site. To each of these mission areas (air defense, for example) correspond specific supporting computer functions. These mission areas functions are dependent upon the type of C^3I system under consideration. There is of course a spectrum of system types ranging from weapon pointing systems to administrative decision support systems. The four main system types are:

(1) weapon/platform control;
(2) intelligence information and exploitation;
(3) tactical battle management automation; and
(4) top-level strategic force management.

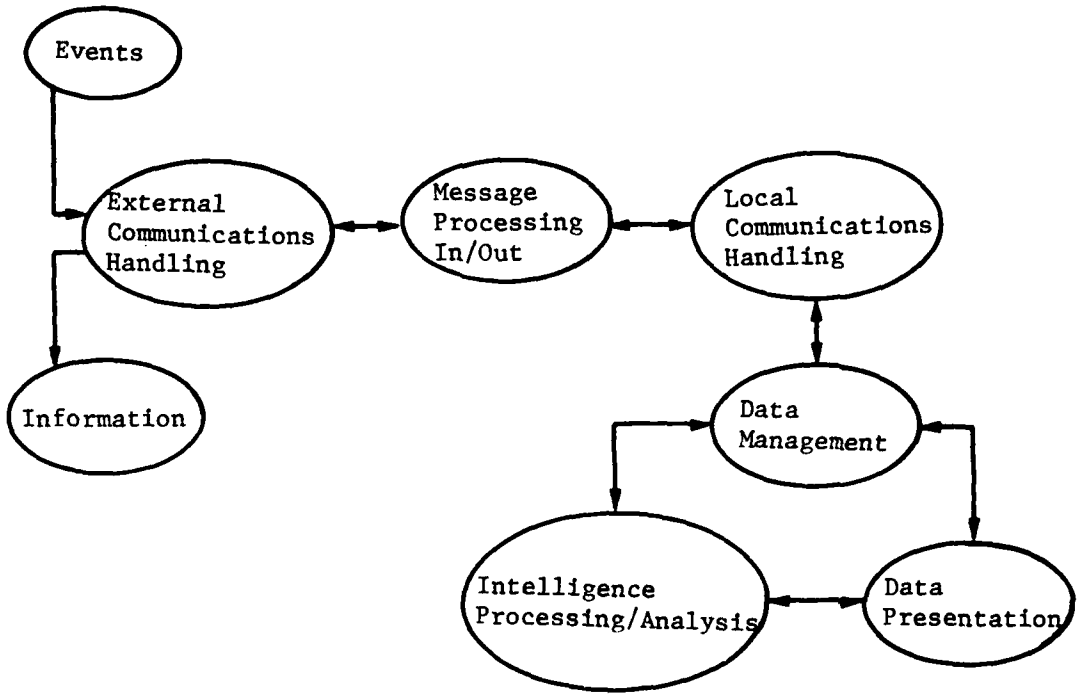


Figure 6.2-1 Kernal-Imbedded Computer Functions

We prepared a structuring or taxonomy of the set of functions which is shown in Figure 6.2-4. There are three mission categories which are partitioned by functional areas, which, in turn, consist of several types of embedded computer functions. Further decomposition is site specific. The mission functions are organization, site, and mission specific. It was against this set of functions and their organization that we began our evaluations.

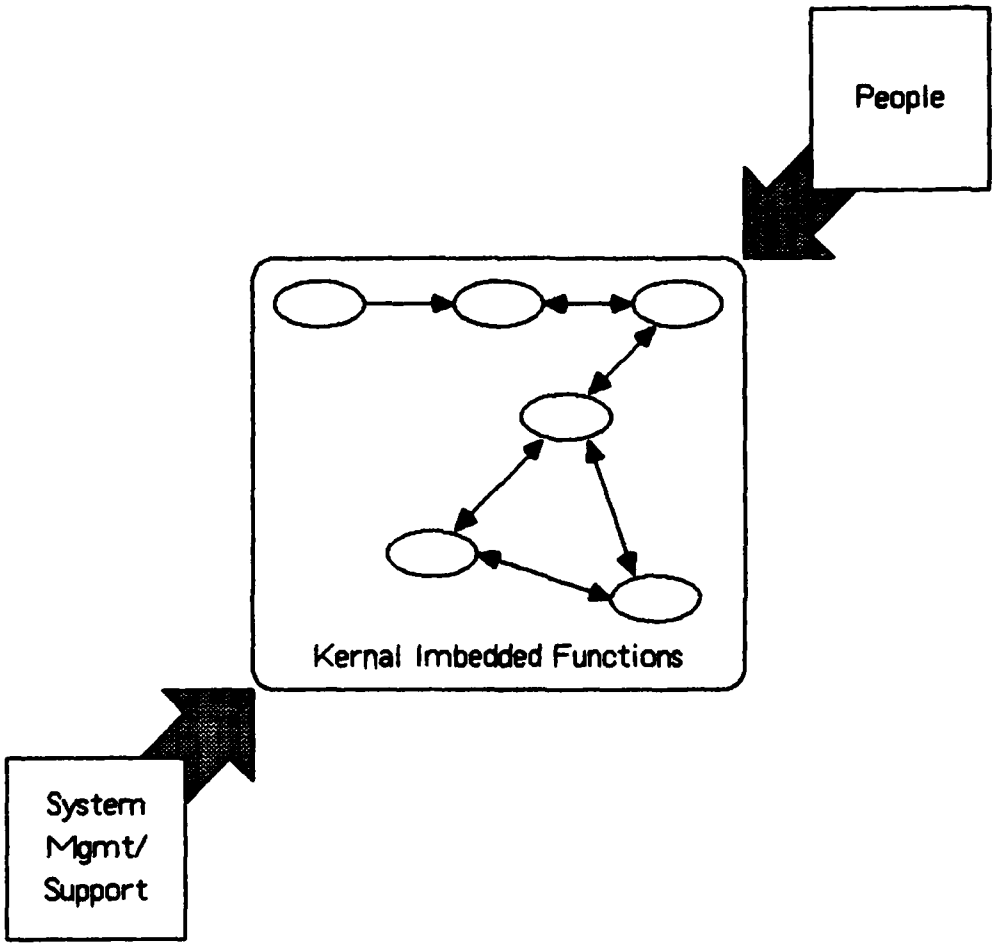


Figure 6.2-2 These functions are made possible by other functions.

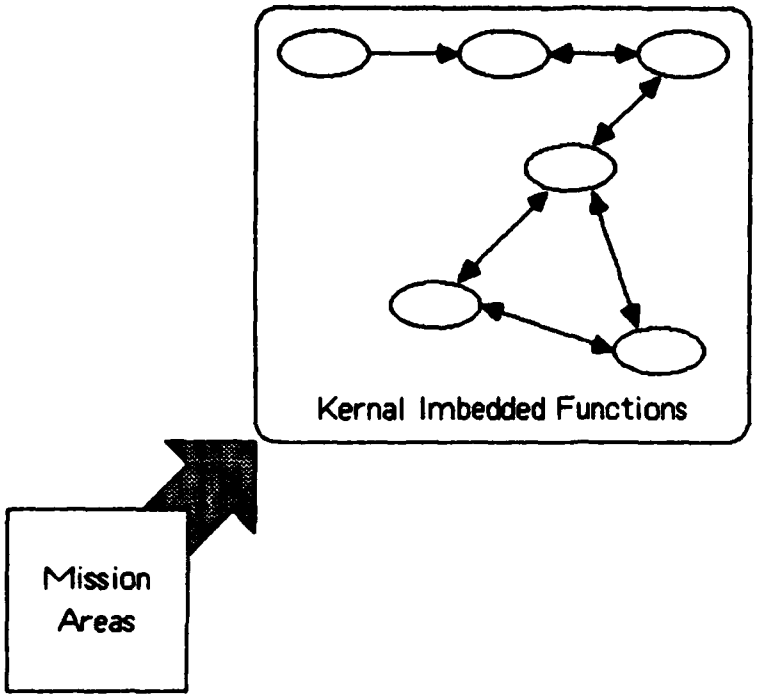


Figure 6.2-3 These kernal functions support mission areas.

Mission	Categories	Functional Area	Embedded Computer Functions
C^3I	Mission Functions	SAC Specific	
		NORAD Specific	
		TAC Specific	
		MAC Specific	
	Support Functions	Communications	Message Handling; External I/F; Internal I/F Crypto/ Sanitization
		Data Management	DEMS; Data Base; Access Utilities; Update Facilities
		Data Presentation	User Interface; Decision Aids
		Intel/Analysis	Tracking Fusion; Decision Aids
	Utility Functions	System Management	Resource Management; Performance Management
		Training	Exercises; Scenarios
		Utility Support	Data Reduction; Backup

Figure 6.2-4 A Taxonomy of Embedded Computer Functions

6.3 Analysis Criteria

Studies

In the course of establishing weighting criteria for these functions, we reviewed some of the relevant recent studies and their recommendations. The Defense Science Board report of July 1978 pointed out the need for C^2

system's focus on the users and the need for its ability to adapt to user needs. It identified the costly nature of software development for C^3I systems, and it pointed out the evolutionary nature of C^2 systems. This last was due to the peculiarly difficult nature of requirements definition on such systems.

Department of Defense Instruction 5000.2 dated March 1980 recognized the unique nature of C^2 systems and recognized that the problem of requirements definition was founded in the support role C^3I systems play to human cognitive processes. It recommended evolutionary acquisition for those systems most tied to cognitive processes. It did not, however, mandate the approach.

The Software Acquisition and Development Working Group's July 1980 report for the Assistant Secretary of Defense for C^3I called for prototyping of C^3I systems and revisions to management and contracting practices. They noted that 60% of the life cycle cost for software is spent after the system is built, not in building it. Much of the 60% life cycle cost is attributable to requirements changes which were unanticipated during initial development. This means that at least some of the cost is attributable to errors introduced in the requirements, design, and development processes. The elements of software development most requiring user input to the development process were judged the most difficult.

The AFCEA C^2 System Acquisition Study final report of September 1982 was an in-depth analysis of the problems/potential solutions of C^3 systems. The study introduced the idea of a Rapid Requirements Definition Capability (RRDC) under the control of the government. This laboratory would be a prototyping and analysis facility to aid generation of better preliminary requirements and better/more timely evaluation of contractors' products. The study also recommended that a system architectural context be established for C^3 systems built around the seven-layer ISO model of local area networks (LAN). The study strongly supported evolutionary acquisition for these systems.

The National Security Industrial Association's Software Working Group's August 1983 report extended the recommendations of the AFCEA report. It recommended discarding MILSTD 483-490 in favor of MILSTD 1679 or SDS. The study identified user interface issues as the proper focus of an RRDC because they are the most visible examples of requirements related to user cognitive processes. The report also showed how evolutionary acquisition is a viable approach based on the OASIS experience. OASIS successfully delivered 22 blocks of functionality and has been judged a management success by the Air Force.

The conclusion of this examination of these recent studies is that user interface and other cognitive process supportive system elements are strong candidates for prototyping.

Selection of Criteria

The potential criteria that were established to compare C^3I functions as subjects for prototyping are:

- o degree of importance to final product;
- o degree of importance within the life cycle;
- o extent of existing tools which support the development of the function;
- o other - experience, personal choice, or literature;
- o some combination of these.

Note that the importance of a particular function varies with the system and its mission. Further, the importance of a function varies with the context of a system's use. For example, a weighting of the importance of functions for an IDHS during an exercise in CONUS versus a weighting of the same system's function when deployed on the second day of conflict in central Europe would be different.

There is also a problem with respect to the definition of importance of a function. One definition would select that function as most important which takes the most effort to properly develop with most risk of error. A second definition would select that function as most important which, if removed from the system or undefined during a given phase, would cause the most total system failure.

Given all of these conditions, we evaluated the functions against a combination of the criteria above. In evaluating the function's importance to the product, the first definition of importance was used. To establish importance within the life cycle the second was used. There were three assumptions. The mission functions were discarded as they are not common kernel elements. Further, they are site specific. Training was discarded as it is not a near-real-time critical function. Utility support was discarded as it must be defined with respect to other functions which are therefore more basic.

6.4 Evaluation of Functions

The first valuation was of the importance of the function within the life cycle phases. This valuation was presented in a rank ordering of functions by phase. Figure 6.4-1 shows the results. Phases are ordered across the top with some explanation at the bottom. If there was equality of importance, then the numerical rankings were assigned equally.

Figure 6.4-2 presents the scorings. Column one is an evaluation of the importance to the final product. Data presentation was most important as the user's satisfaction with the system is founded on how he interacts with it. The importance by phase is based on the previous figures' results. Figure 6.4-3 shows the way the next column was achieved. We removed the phase scores for integration and test as we have already decided that prototyping will best support the study, requirements, and preliminary design phases. The final ordering has data presentation functions as the most important.

Study	Requirements	Design	Implementation	Test & Integration	Deployed Product
Mission Functions	Comm (Subsystem) Data Base	Data Mgmt Intel/Analysis Data Presentation	Data Presentation System Mgmt Intel/Analysis	System Mgmt Data Mgmt	Data Presentation Intel/Analysis Comm
Comm (System)	Data Presentation Intel/Analysis System Mgmt	System Services Comm	Comm Data Mgmt	Data Presentation Intel/Analysis Comm	System Mgmt Data Mgmt
Tend to deal with the system as a whole.	Info flow assuring system integrity is paramount. Useability is secondary.	Difficulty of design is basis for this. The core of a design is the data mgmt system.	Displays/ picture dynamics key.	This phase involves comparison to rqmts (paper). The user-oriented functions are generally incompletely exercised at this time.	User validates system through MMI. Other systems are transparent.

Figure 6.4-1 Examination of Priority Based on Life-Cycle Phase

	Importance to Final Product	Importance by Phase					Have vs Needed	Literature	Total Raw	Weighted Total	Final Order
		S	R	D	I	T					
Communications	3	1	1	5	4	5	4	3	26	15	2
Data Mgmt	5	3.2	2	1	5	2	1	4	22.2	15.2	3
Data Presentation	1	3.2	3	3	1	3	2	1.5	17.7	13.7	1
Intel/Analysis	2	3.2	4	2	3	4	3	1.5	22.5	15.5	4
System Mgmt	4	3.2	5	4	2	1	5	5	29.2	26.2	5

Figure 6.4-2 Numerical Ordering of Functions

Functional Area	Means	Weight
Communication	Residue S/W	4
Data Management	None	1
Data Presentation	Test Beds	2
Intel/Analysis	Residue (Some)	3
System Management	Commercial	5

Figure 6.4-3
What means currently exist to support prototyping in each area?

Therefore, we conclude that data presentation is the key embedded computer function based on abstract reasoning. It is sensible to ask if this conclusion will stand up under examination of real systems or if it is validated by other data. In particular, does a real system show differences between first presentations (ranked data) and communications (second ranked)?

We studied National Cheyenne Mountain Center (NCMC). Part A of Figure 6.4-4 shows the top level functional decomposition of NCMC. Part B of the figure compares graphic display types required to support NCMC mission areas to output message types required for the same support. The numbers occurring in the columns of Part B are the number of different output display types compared to the number of different message types. This data was obtained from an ADCOM prepared functional description document. The figure makes the case that in constructing the NCMC system, there would be significantly less software to support message preparation than to support display preparation software. Displays in this case are outputs to the user which inform him of situational elements. Comparison of output message preparation difficulties to displays has some basis.

NCMC	
Mission Functions	- Missile Warning - Space Defense - Air Defense - Battle Staff Support Center
Support Functions	- Communications - MMI - Intelligence
Utility Functions	- Verification - Performance Mgmt - Resource Mgmt - Historical Data System - System Exercises - Recording - Data Reduction

(a)

	Mission	Missile Warning	Space Defense	Air Defense	Battle Staff Support Center
MMI	Graphics Displays	Classified	133	0	0
	Hardcopy		104	8	12
	Tabular Displays		0	14	8
	Alarms		55	5	1
	Display Category		0	9	2
	Display Special		0	3	1
Communications	Output Messages		89	4	6

(b)

Figure 6.4-4 Close Examination of a Particular System

The current prioritization of function types as candidates for prototyping are:

- Data Presentation
- Communications
- Data Management
- Intelligence/Analysis
- System Management

In the course of evaluating these functions, several past or ongoing C^3I systems or their components were studied. These are listed in Figure 6.4-5. In all of these except 427M and CSSR, data presentation is either the key concern or the number two concern.

- SPADOC	- Original TFCC
- SPADOC 4	- GACC
- CSSR	- Constant Watch
- 427M	- ITSS
- CCPDS Upgrade	- OBU
- OASIS	- Various Classified Efforts (Sanitized Mode)

Figure 6.4-5 Systems We Have Examined

6.5 Matching Functions to Prototyping Approaches

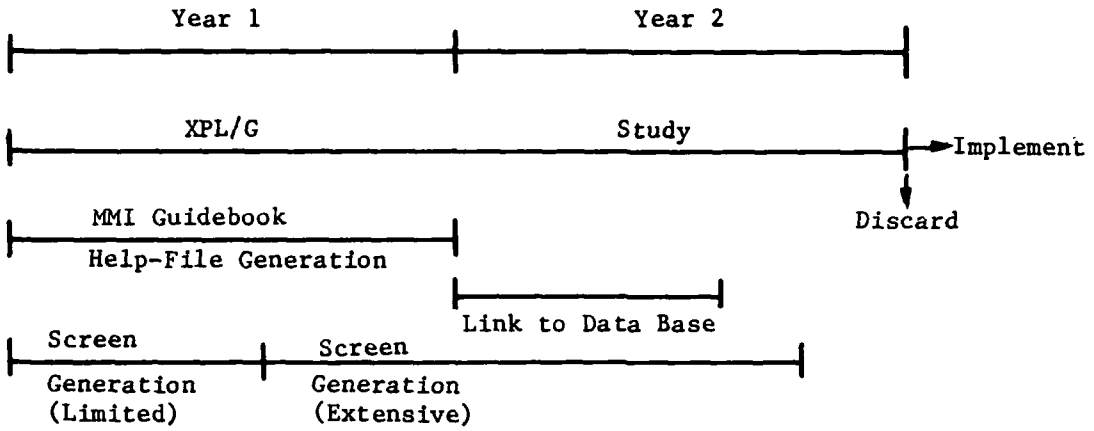
Prototyping approaches were evaluated with respect to C^3I systems and their components, as were specific tools as instances of these approaches. We have been able to select the most appropriate approaches to each of the top three C^3I functional areas.

Data Presentation

In terms of formal description languages, Mallgren's XPL/G is the most powerful and general. It has yet to be instantiated for C^3I applications. A few years' effort building up C^3I specific constructs in the language would be enough to judge its further suitability as a tool for C^3I system development.

In the area of SPDA approaches, hierarchical or guidebook type models exist. They are mechanized to a limited extent. Smith's MMI definition guidebook work at Mitre is extensive. This would have to be mechanized as a large help file in conjunction with a carefully-thought-out template system. Such a set of hierarchical templates or menus would guide a user in the population of a data base which modeled MMI. This data base would then have to be linked to a functional testbed system to allow exercising of the prototypes. This is a promising area. Section 6.7 discusses this further. Functional testbeds have already had impact on C^3I systems through prototyping. Both Ford Aerospace and Martin Marietta Denver Aerospace teams used prototyping during the SPADOC 4 definition phase to refine operational requirements. No work needs to be done to apply those systems today with effect. A system such as the Denver Aerospace C^3SL is more costly to develop than TRW's GIDS/FLAIR. One can obtain more dynamics in the resultant displays by expending more money. These systems would profit from coupling to a hierarchical MMI model as described above. The improvement would be in a higher confidence and more complete MMI definition as input to the testbed. Use of forms or screen generation techniques are likewise mature for limited application in C^3I system developments. Effort must be made, however, to provide complex graphics as a part of existing forms packages.

Figure 6.5-1 shows the timelines to exploit these different approaches. The functional test bed approach can be used today, and most of its drawbacks could be improved by coupling it with the guidebook work. Therefore, this combined approach is recommended.



Functional Test Beds
- Denver Aerospace C^3SL
- TRW GIDS/FLAIR

Figure 6.5-1
Exploitation Timelines for OSI Prototyping Approaches

Communications

Formal definition language based systems do not exist for communications system elements of C^3I systems. Within the SPDA approach C^2SAM by BETAC/LOGICON is an important tool. It models organizational structure and informational requirements for existing C^2 systems and will tell a user what messages need be received for a newly defined system. Necessary extensions to C^2SAM are in the area of improving its user access and in coupling it to other packages/data bases. The Denver Aerospace SMARTS is a large C^2 message traffic generation system falling within the scenario generation/test harness approach. When finished it would provide extremely detailed operationally defined message system prototypes. Its use in rapid prototyping would be very limited, however, by the size and complexity of the system. The recommended approach is through use of the C^2SAM.

Data Management

While much work has been performed in the area of data base system (structure and management systems) development methodology, there has been little work on prototyping of these systems. One approach is built around DEC's DATATRIEVE. The use of DATATRIEVE to produce different logical views of the same physical structure would support prototyping as follows:

> A command language would map a set of user defined views to a DBMS. This would invoke DATATRIEVE and map the logical view to applications processes such as graphics displays and simulations. The applications would be hidden from the "initial" existence of the data base system they were using.

We have used DATATRIEVE at Denver Aerospace to prototype portions of a NAVY C^2 system. The use of Prolog has great potential. It is well known that any database structure (hierarchical, relational, or network) can be described with a relational model. This can be a first normal form expression of the schemas. Both management systems, access schemes, and data structures can be easily described in Prolog's (first order predicate calculus constructs).

6.6 Programming Environments

The concept of a programming environment (PE) has received much attention since Winograd's 1974 paper. Basically, PE's are integrated sets of software tools designed to support a software builder in constructing programs in a particular language. These tools act as an approximation of an assistant standing over the builder's shoulder and aiding him in managing the complexity of the software design task. Common examples of PE's are UNIX and INTERLISP.

This notion of a design support environment has some utility as a means of rapid prototyping. The INTERLISP environment especially has been designed to support evolutionary or experimental programming. This notion is the current way LISP-based programs are developed given the difficulty of specifying requirements for artificial intelligence software.

Role of PE in Rapid Prototyping

Proponents of the PE as a rapid prototyping tool subscribe to the quick build philosophy. By decreasing the time spent in detailed design, code, and checkout, judgments may be made about requirements. Appealing to our earlier discussion (including Figure 4.1-3), this means the approximation of the functionality of the software is not an approximation, but rather fully functioning software. The constituents of these environments are shown in Figure 6.6-1. Some programming environments are listed in Table 6.6-1.

There are two circumstances in which the PE approach has utility. First, if the target language for the implementation is the focus of the PE and if the PE is hosted on the target ADPE, then the required software may be built in a friendly environment. We can expect to spend less time in detailed design through checkout in such an instance. Further, requirements instability can be expected to have less effect on these later life cycle phases due to the presence of tools to manage change. Major requirements issues on the A or B1 level will still have a major destabilizing effect however. The importance of using the PE hosted on the target ADPE relates to the importance of MMI functions for the C^3I systems. A major return of using PEs is quick check-out. If intermediate steps are needed to port the developed code to try it out on the target display suite, much of the environment's utility may be lost.

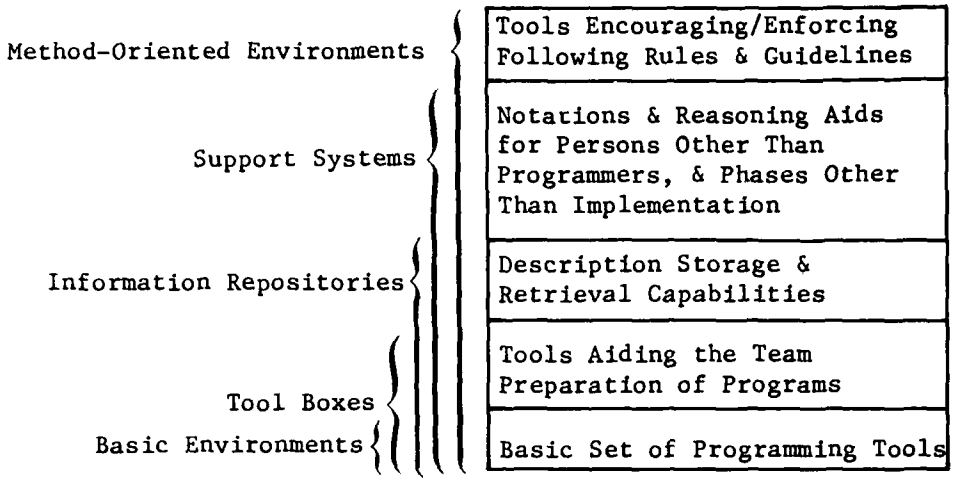


Figure 6.6-1
Programming and software environments may possess varied capabilities.

The second instance where use of a PE would be of aid is when requirements are undetermined, yet software must still be constructed. This can occur during the study portion of a C^3I acquisition, before full scale development (FSD); or it can occur when artificial intelligence software is being built. The PE would support experimental programming. This can take place either when there is no particular software deliverable required of the effort, or when there is no other choice. This means that we were forced to accept the risk that a non-useable product was obtained.

Programming environments therefore have some ability as means for rapid prototyping on C^3I developments. They support the quick-build approach to prototyping by accelerating the later phases of software construction. They aid in evaluating requirements through mini-lifecycle-based feedback and really do nothing for expressing requirements in structured form. Rather, they support the direct processing of English-language-based requirements. Another drawback is the lack of C^3I specific information in the tool base of these environments.

Table 6.6-1 Some Programming Environments

Name	Language Supported	Supplier/ Reference	Experimental vs Commercial	Methodology Enforced
UNIX	C, PASCAL	Bell, Berkeley	C	No
SMALLTALK 80		Xerox	C	None
INTERLISP-D	LISP	DARPA, BDM, Others	C	None
MENTOR	PASCAL	INRIA	E	None
TOOLPACK	FORTRAN & a Command Language	L.J. Osterwell	E	None
JOSEPH	Reqmts Language Pharoh Model Language - OASIS	W.E. Riddle	E	None
DREAM	Special Notation	W.E. Riddle	E	Top-Down
GANDALF	ADA	Carnegie-Mellon University	E	Some

6.7 Object Oriented Programming

Xerox Corporation's learning research group from Palo Alto, California has done extensive work with children to investigate natural ways of communicating with computers. Out of this research came the object oriented programming language Smalltalk, that was developed in 1972 and has since undergone numerous iterations. Smalltalk is a simple but powerful concept and is more limited in scope than current dialects of LISP. Under the LISP machine project begun in 1974 at the MIT AI laboratory, the Smalltalk concept was altered and extended to become the FLAVORS system.

Although the system contains many Smalltalk constructs, it also permits multiple inheritance, a characteristic that Smalltalk did not have. Multiple inheritance is the ability to inherit characteristics from classes (similar objects) that are hierarchically unrelated. Another major difference is that Flavors is a fully integrated extension of Lisp.

Object-oriented LISP programming is concerned with modeling the behavior of real world or abstract objects. It depends upon four key concepts: objects, instances, methods, and messages. An object is an encapsulation of data structures and of the functions or methods that operate on them; an object can be sent a message requesting that it enact a method.

Early implementations of object oriented programming were designed so that objects fell into strict hierarchical patterns with levels successively more specialized than earlier ones. In Zeta LISP the generic object classes are called Flavors, which along with the methods associated with them are defined by the programmer. The Flavor system differs from the earlier implementations because it allows non-hierarchical inheritance; thereby providing the means for building arbitrarily complex Flavors while retaining the advantages of modularity and maintainability.

Thus, some flavors are designated by convention as base flavors, others as mix-ins. The latter add particular features to other flavors. Users may define flavors by combining base and mix-in flavors as needed. Defining methods for these new flavors can over-ride, augment or modify the methods from the component flavors. This non-hierarchical definition facility distinguishes flavors from hierarchically dependent systems like Smalltalk, and gives Zeta LISP programmers additional power and flexibility. The RPS system developers, especially the system programmers, may well find tremendous advantage in the use of Flavors to build complex C^3I specific models. These models would be executable because of being implemented in a Flavors-like system.

Flavors represents generic objects. For example, a generic ship with its description could be a Flavor. An aircraft carrier with the addition of its unique and non-generic descriptors would be an instance of the Flavor "ship." The process of going from the generic to the specific is called instantiation. Such instantiated Flavors are manipulated by sending messages that request specific operations. Since the procedures are already contained within the object, it responds by performing the operation requested.

There are two areas in the RPS where such object-oriented environments allowing the sending of messages between objects having the ability to operate may be useful. The first is in extensions to the RPS executive or to the RPS specification component. The object-oriented environment concepts, specifically something like Flavors, could also be used as a means of structuring macros in using the RPS system. For example, as prototypes were developed, should they be stored in a Flavors environment, we could use a particular prototype definition as if it were a function itself and therefore build up in a modular fashion a prototype of a much more complex system--or be able to make use of the prototype libraries--aggregating and de-aggregating them on a very high level. This would support, after some use had been made of the prototyping system, the ability to use these large detailed prototype definitions as functional components. This would in turn allow examination of the system that we were studying at a level of detail closer to the detailed design phases of the life cycle. Certainly this is not a replacement for code generation systems and other methods of automatic generation of detail design from preliminary design or preliminary design from validated requirements. However, it would provide a way of exercising, in a fairly thorough manner, part or all of unvalidated conceptual designs.

7.0 RELATIONSHIP OF PROTOTYPING TO MILITARY STANDARDS

Our study thus far indicates that only minimal changes need be made to relevant Air Force Standards. We believe this because the language of the standards implicitly provides for the use of a rapid prototyping tool in system acquisition. The word tool is used here in the sense of something used to augment or facilitate our existing process. Rapid prototyping is a tool augmenting the system analysis process. It does not imply any fundamental addition to the process but rather an improvement of the validation and analysis activities already embodied in the standards.

Another way to consider this is through a modular view of the acquisition process. The standards MIL-STD-483 and AF-800-14 establish the high level description of the process. Validation is a subfunction of the overall process. By viewing rapid prototyping as one of many possible implementations of the high-level concept of validation, it is clear that explicit language in the high-level standard unduly restricts choice of mechanization in the process. Rapid prototyping implements subfunctions of the validation portion of the acquisition process.

Rapid prototyping, however, does implement a validation activity in a significantly unique manner, so the standard should reference it to some extent. Reference should be limited to simple textual addition and need only be included in AF 800-14. MIL-STD-483, which we also considered, and MIL-STD-490 do not deal with issues truly relevant to prototyping. MIL-STD-490 deals with program-peculiar item specifications and is much too detailed and rigid to be applied to rapid prototyping. Conversely, rapid prototyping in no way applies to specification practices in the sense that MIL-STD-490 deals with them. Prototype software is not deliverable and, while needing management by the developer, should not be subject to 483. We feel that it definitely should not be subject to 483. The organization directly involved, contractor or acquirer, should control the configuration of a prototype according to their internal standards for nondeliverable software.

In summary, prototyping is an adjunct to the normal analyses occurring during system development. It happens that it may involve software development. In general, prototype software does not migrate into the deliverable software; rather, insights from the process of prototype software development influence the process of deliverable software development.

The tables in Appendix A3 enumerate the changes we are considering recommending to AFR 800-14 as a result of our current review of the standard. Column one lists the specific section of the standard; column two, acquisition process topic; column three, the issue in regards to that topic; and column four, our preliminary recommendation.

8.0 LESSONS LEARNED IN DENVER AEROSPACE TESTBED

Our investigation juxtaposed the desired rapid prototyping capability and existing prototyping techniques. We have learned several lessons from this side-by-side comparison. Four in particular have guided our study and provide direction for future effort.

First, we learned about the nature of the rapid prototyping 'tool' to be developed. The state of the art makes it clear that no single technique or small group of techniques will satisfy the need, even within the restricted domain we are addressing. Only the proper rapid prototyping environment can provide the full complement of capabilities being sought. The environment must tie together a range of tools, specialized and general purpose, in a coherent package.

Second, we learned about the nature of this investigation. We seek to develop a system to augment the development of other target systems. This development system is based on the assumption that it is better to 'try before you buy.' Although this effort is restricted to a particular set of target systems, the prototyping concept itself has much greater generality. In fact, the concepts to be embodied in the rapid prototyping environment apply to the development of the environment itself. To validate these concepts, we must use what knowledge we have of them now, even as we develop a system to exploit them. To build a rapid prototyping system, one must prototype that system.

The third lesson is that prototype development is a microcosm of general system development. Usually the only distinguishing attribute of a prototype system is that it is less complex than its counterpart production system. In some cases, not even this is true.

Lesson four says that we should borrow as much as possible from the general system development environment. This follows from lesson three because what is good for system development should be good for prototype development. It also derives from another observation. Prototyping exists so that results of exercising prototypes can influence the requirement and design of production systems. Therefore, it is vital that communication be established between prototyping activities and activities that use the results of prototyping. By sharing the system development environment, formal and informal flow of information can be established.

With these ideas in mind we have investigated the various tools and techniques that seem likely candidates to make up the rapid prototyping environment. In the process of studying tools and techniques we are identifying a set of testbed requirements. Also, in the process, we have brought together many of the components of the testbed in true prototyping fashion.

8.1 Context of the Rapid Prototyping Testbed

The process of developing software-dependent systems is part discipline and part art. In this respect it is not unique in the world of system development and engineering. The software dimension, however, shifts the process further into the world of art. By discipline we mean structured methodology that can be applied consistently to produce desired results. Art

refers to the use of style and instinct to achieve results. The "artfulness" in the development of software derives from software's inherent flexibility, its abstract nature, and its relative infancy as a discipline.

Software exists at several levels; viewing it at the proper level reveals the inherent flexibility. At the lowest level we can look at software as a static bit pattern in some storage medium. Such a bit pattern in execution on a computer can be thought of as a process. At a higher level, software is seen as a description of processes. This process-description level characterizes software in terms of language constructs. Here the flexibility of software becomes apparent. Languages, even primitive ones like programming languages, can express an infinite number of ideas. Almost any conceivable process can be expressed. Furthermore, the same process may be expressed in thousands of different ways.

The perspective of software as a description of processes also reveals the abstract nature of software. A description is static. Processes are dynamic. Understanding software requires mental abstraction between the description and the process. We never really "see" the process. We interact with its software, we see its inputs and outputs, but we can only perceive in our abstract and personal way the executing process.

The science for dealing with abstract software processes is relatively young. In absolute terms, we have been developing software for less than three decades. In that time, software has undergone a divergent evolution. Software has been applied to a vast array of needs with an equally vast array of techniques. Lagging behind the pressing need to apply software are efforts to develop an encompassing view and supporting discipline. This is not to say that we do not have any discipline in the software development world. The life cycle of software is generally understood, and there are many techniques to support it. However, we are far from consensus on many issues. In any case, the mainstream of software development has not applied many of the available solutions because they are not mature enough to be smoothly integrated into the ongoing process of software development.

These factors paint an unsettling picture of software development. It is one of a poorly mapped and hostile land we must cross to get from user's needs to final solutions. To cross it we must rely on a collection of guides, pundits and wizards. Understandably, the user, acquirer, and developer would like to replace a few of the wizards with a good road map. They would like to see the science of software development mature.

The purpose of the preceding discussion is to establish the context of our efforts to create a testbed for rapid prototyping concepts and techniques. Figure 8.1-1 illustrates the context. Our effort is part of a larger effort to create a disciplined software development environment. We are assembling a test bed in which to develop a prototype of the rapid prototyping environment. The rapid prototyping environment is intended to augment the C^3I system software development environment which is part of the general software development environment. All of these reside in the larger context of system development.

This context is important to remember because in planning the rapid prototyping environment we must plan to integrate our tools with other tools designed to improve the software development discipline.

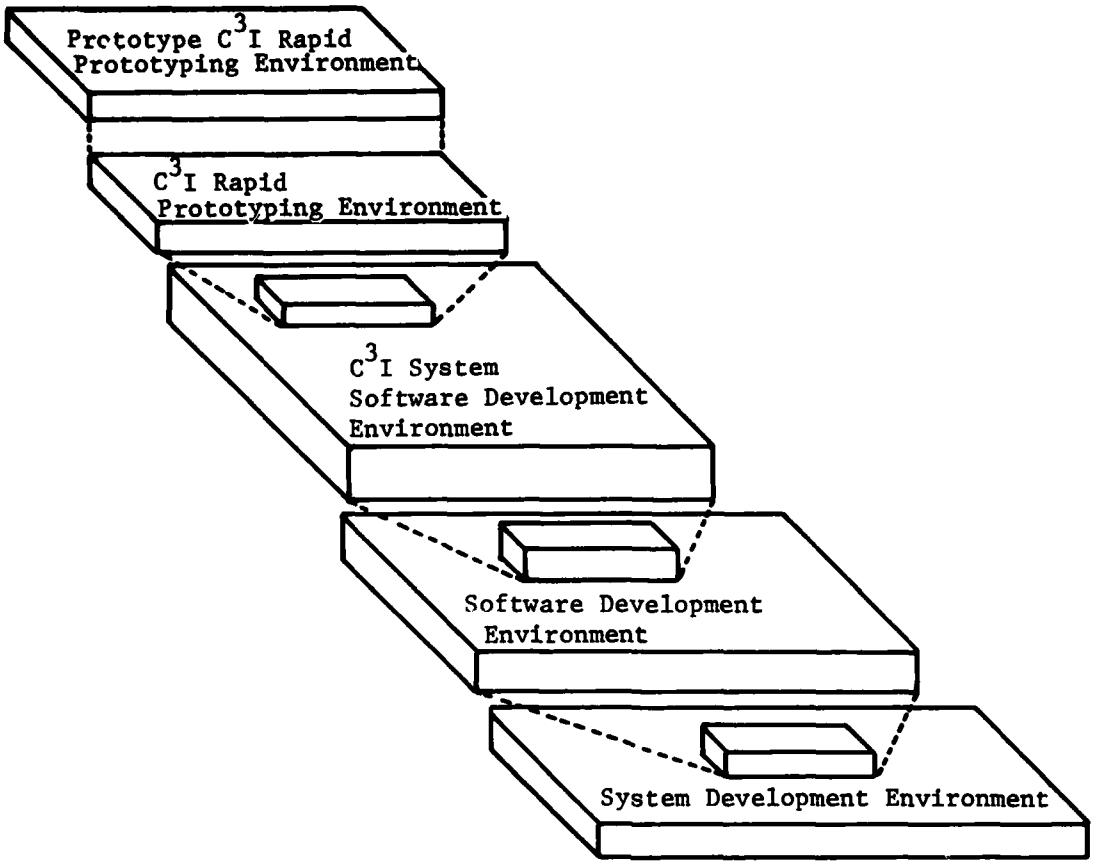


Figure 8.1-1 The Context of Our Effort

8.2 Prototyping's Relationship to Other Software Development Tools

In section three we discussed the specific problem of developing C^3I systems. Our conclusion is that most problems arise from inadequate definition of requirements for those aspects of the system directly supporting the site user's cognitive processes. In section four we advance rapid prototyping as a solution. In section five we outline the various approaches taken to rapid prototyping. The preceding section establishes a view of rapid prototyping as one aspect of the maturation of the software development discipline. From this basis we can now discuss the relationship of prototyping tools and concepts to other software development tools. To describe the relationship we need to talk about the issues addressed by software tools, the spectrum of available tools and prototyping's place in that spectrum. Then we can isolate common aspects of prototyping and other tools.

The software life cycle is generally understood and agreed upon. It provides a conceptual outline of the issues to which software tools have been applied. Figure 8.2-1 shows Denver Aerospace's view of the life cycle and layout of the issues relevant to phases of the cycle.

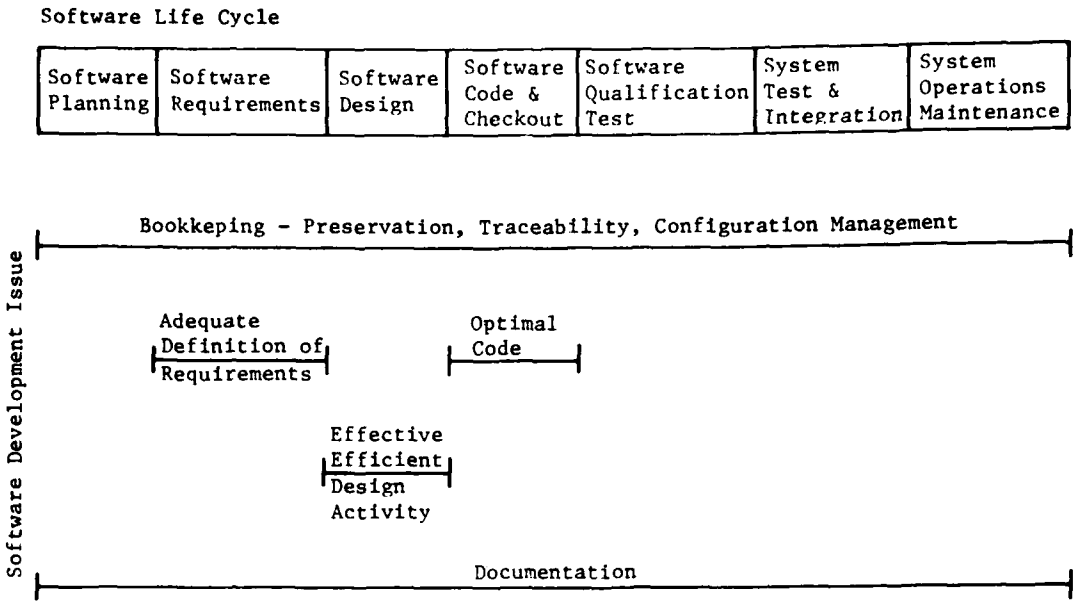


Figure 8.2-1 Software Development Issues Addressed by Software Tools

Software Development Issues

Bookkeeping is a practical issue that spans the entire life cycle. As a system progresses from its conceptual phases into detailed design and eventual operation, we learn a great deal about the system. The information might be in diagram and drawings, electronic storage media, paper documentation or in the engineer's mental conceptualizations. The problem is to preserve this accumulation of information because it is, literally, the system at any given time.

Along with preservation of information, there are the parallel concerns of traceability and configuration management. Traceability ties individual, detailed bits of information to higher level concerns; for instance, "What information about design relates to a particular requirement?" At a lower level, "What data implements a design?" We have to trace these kinds of information in order to understand how the system needs are being answered. Also, we must know what parts of the growing system are affected by change in another part. Configuration management is concerned with the present state of the system and its description. Keeping track of these types of questions is essentially a bookkeeping problem.

The problem of adequate definition of requirements was discussed in Section 3. The general issue in software development is how to apply discipline to the requirements analysis process, thereby ensuring the correctness of the results. Given stable requirements, what is the proper design process that will resolve the requirements? First of all, the activity must produce effective designs that satisfy completely the requirements. Then we need to concentrate on making the design activity efficient in terms of optimizing resources used. The concern in the code checkout phase is that the resultant code is optimized for several factors. Code must be easily

modifiable, maintainable and easy to construct. Documentation is the tangible aspect of a software system. It allows us to understand, use, maintain, and modify the system at any time. The issues for a rapid prototyping system are how to produce documentation in a timely and efficient manner and how to ensure agreement between the actual system and the documentation.

Spectrum of Available Tools

Figure 8.2-2 illustrates the relationship of some software tools to the life cycle and required document milestones. Figure 8.2-3 relates these same tools to the software development issues enumerated earlier. The relationship of each tool to each issue is rated one, two or three. A rating of three indicates that an issue is the primary concern being addressed by the tool; two means that the issue is an important concern; and one means that an issue is only of indirect concern.

Programming Environments

In Section 6.6 we developed the concept of programming environments which are particularly applicable to rapid prototyping. These type of environments are a few of many different environments. The primary concern of programming environments is optimal coding. But these environments also address design and documentation issues. The system semantics and data structures of language in an environment can be designed to support modular maintainable designs. Programming languages syntax can contribute to self documentation of code.

Environment is not strictly limited to language. The tools to develop code and process it also form the environment. Editors, Linkers, Debuggers, Dynamic Program Interpreters, and others are refinements that increase the overall efficiency of a programming environment.

Graphic Interpretation of Program Structure (GRIPS)

GRIPS represents a class of tools used to graphically portray the control structure of program source code. GRIPS is based on a type of control diagram known as "Visual Control Logic Diagram" in Denver Aerospace terminology. Another name for this representation is Naisse-Schneidermann type charts.

This tool operates on pseudo-code description of actual code. It processes the pseudo-code producing diagrams of the control structure. The result is a more understandable representation of the sequence, decision, iteration and procedures in a unit of code.

Tools of this type are primarily concerned with documentation. The diagrams are required for most DOD projects. GRIPS gives an automated means of maintaining diagrams of code. More importantly, it helps keep code and diagram in agreement. In the absence of an automated tool, the documentation often does not keep pace with changes in the code because preparation of the diagram consumes too much time. With GRIPS a simple editing of the pseudo-code is all that is required to update the diagram.

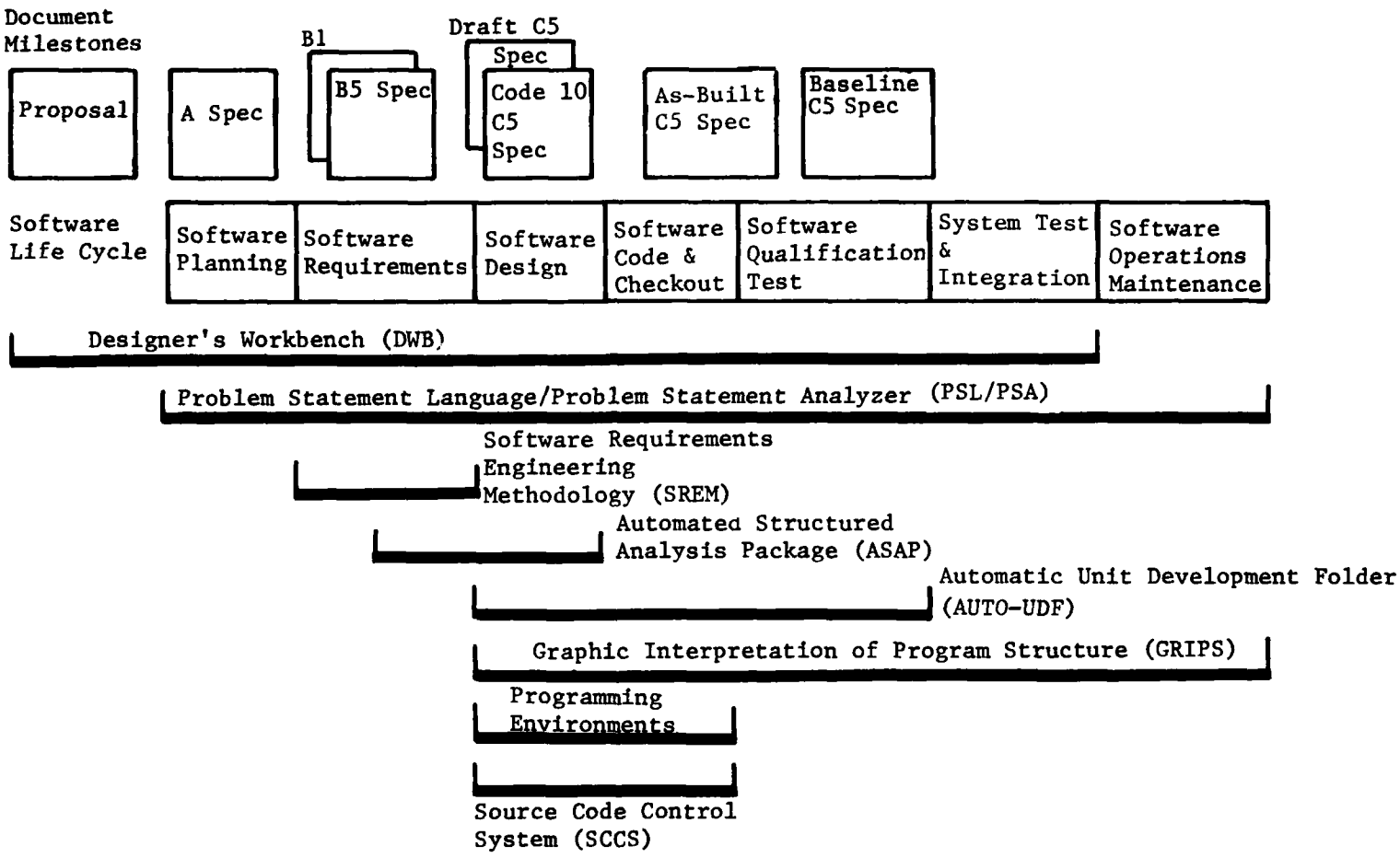


Figure 8.2-2
Relationship of Software Tools to the Life-Cycle and Document Milestones

Tools	Issues: Bookkeping	Adequate Definition of Requirements	Effective, Efficient Design Activity	Optimal Code	Documentation
SCCS	3	1	1	1	2
Programming Environments	1	1	2	3	2
GRIPS	1	1	1	2	3
AUTO-UDF	2	1	1	1	3
ASAP	2	1	3	2	2
SREM	2	3	1	1	2
PSL/PSA	3	2	2	2	3
DWB	3	1	1	1	2

Rating: 3 Primary Concern; 2 Important Concern; 1 Indirect Concern

Figure 8.2-3 Software tools relate to software development issues.

Optimal code is also an important concern with these types of tools. Graphic representations aid the programmer in understanding the abstract processes being developed. A picture is worth many words.

Automatic Unit Development Folder (Auto-UDF)

This tool was developed at Denver Aerospace using the Unix environment and elements of the Source Code Control System (SCCS). It implements internal standards for documentation of units of code. An individual programmer using the tool is provided a template which she/he must complete. The template aids the programmer in documenting schedule, test, design and the actual implementation of coding units. The tool provides the services on-line, thereby reducing the effort of storing, producing and communicating this information.

Software Requirements Engineering Methodology (SREM)

SREM is a methodology and support environment for refining software requirements from system requirement documents and expressing and validating these requirements. The tool was developed for the Air Force by TRW. It was evaluated over a twenty-month period by Denver Aerospace under contract to the Air Force. The evaluation team's conclusion was that the methodology was valuable and the support tool very helpful in expressing and storing requirements analysis information. However, they found it difficult to express certain real-time and near-real-time constraints.

Part of the support environment of SREM was designed to generate functional simulations of the system based on the requirements analysis results. It was found that this aspect of the SREM was cumbersome to use and

that the simulations produced were not of very great value. The primary concern of the tool is adequate definition of requirements. In this regard, the data base and the associated methodology do provide significant help to the activity. The language of the data base is well suited to C^3I in particular. SREM also provides help for documentation and bookkeeping. This is simply a function of SREM's data base.

Problem Statement Language/Problem Statement Analyzer (PSL/PSA)

This is a very ambitious tool that concerns itself with each of the issues. Its possible usage extends over the entire life cycle. It is centered around a data base which accumulates information. The information is expressed in the PSL part of the tool. The PSL is designed to be processed by an analyzer. Bookkeeping is addressed through the expression and storing of information in the data base. Documentation is provided through formatted reports derived from the data base. The other issue we addressed through the analysis of the stored expression. In a limited way the analysis indicates the logical completeness and consistency of the specification (Problem Statement) at any time. It is limited in that it does not tell the developer, "yes you have stated the problem correctly", but rather "yes you have stated a well-defined problem".

Automated Structured Analysis Package (ASAP)

ASAP primarily supports the design activity. It is essentially an automated Yourdon data-flow analysis support tool. It allows the expression and storing of process descriptions and data definition. Requirements may also be expressed and stored for the purpose of linking them to detailed process designs and data definitions. ASAP will examine the described processes and data to determine how complete and consistent the design is in terms of data usage between processes.

Designer's Work Bench (DWB)

Designer's Work Bench is a Martin Marietta tool which is centered around a data base and language generating tool for the data base. The approach taken by DWB represents a coalition of lessons learned on other comprehensive life cycle tools. PSL/PSA, SREM and ASAP are all centered on data bases also. However, these tools have attempted to provide a measure of expression encompassing all possible needs. In addition, they have a built-in world view of how to support the lifecycle activities. DWB's approach is to provide a meta-language in which to instantiate various syntaxes used to express requirements, design, pseudo-code, data-definition or whatever is necessary. Having done this, analysis and documentation are supported through the use of the data-base query facilities. Where simple data base query falls short, one may simply extract the information from DWB in the proper form to be processed by a specific tool. For instance, ASAP process description might be stored in DWB. This would allow automated YOURDON analysis to be performed on data stored by a comprehensive life cycle tool. Pseudo-code for GRIPS could be stored and linked in the data base to the relevant code units.

The possibilities are numerous. That is the point. By focusing on the bookkeeping activities of the life cycle at a meta-language level, DWB in effect addresses nearly all the life cycle issues. What it does not itself do

can be provided by other tools without changing all of the bookkeeping practices.

Prototyping in the Spectrum of Software Tools

So far, we have related the life cycle to software development issues, the life cycle to software tools and software tools to issues. Figure 8.2-4 relates rapid prototyping to the life cycle. It applies to the time from before the life cycle even begins to the end of the software design phase. Early on in the cycle it would be used to validate high level requirements. In the design stage, prototyping would be used to develop detailed performance information about design.

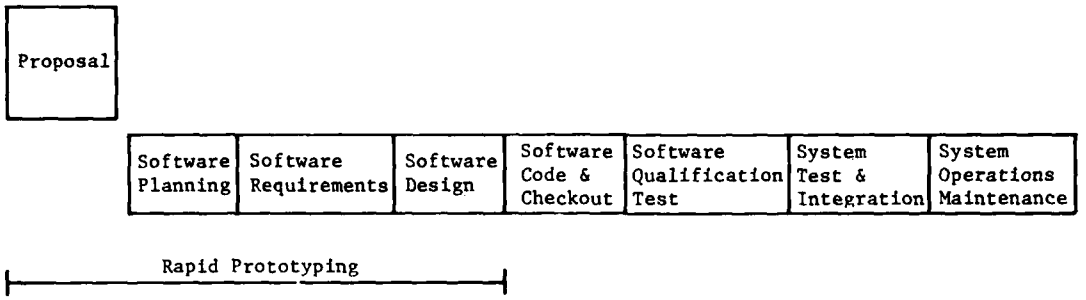


Figure 8.2-4
Rapid prototyping relates to the life cycle.

We are constrained to apply rapid prototyping to critical high payoff aspects of embedded computer functions in C^3I systems. We have concluded from our studies that the greatest payoff results from the stabilization of requirements at the highest levels. Therefore, we will not focus on applying prototyping to the design phase. To place rapid prototyping in the spectrum of software development tools we put aside for the moment prototyping at the system level. This leaves the Software Requirement phase and the adequate requirements definition issue as the intersection of prototyping and other tools.

Referring back to section 4.1 and Figure 4.1-2, we subdivide requirements definition activity into subactivities. The three subactivities form a cycle which results in validated requirements. The three subactivities are identity, express/represent and evaluate. In Figure 8.2-5 DWB, SREM, PSL/PSA and ASAP are related to the cycle. DWB applies only to the expression of requirements. The other three tools contribute to both the expression and evaluation. Rapid prototyping applies to the entire process. It applies to identification in the sense that it exposes requirements; expression in the sense that it gives tangible form to the requirements; and evaluation in the sense that it provides a means of measuring the requirements being prototyped against real world scenarios.

Common Aspects

Rapid prototyping and other software tools have two apparent aspects in common:

-- Application to adequate requirements definition,
-- Interaction with system information data base.

Another aspect contributes to the relationship. In Section 5 we talked about the various approaches to prototyping. One approach was the "build it twice" philosophy. It we view prototyping approaches on a scale ranging from those that implement the system completely to those that mimic only the high level functions, we can see prototyping as a microcosm of the system development environment. It has its own requirements, implementation schemes and life cycle. Software tools apply to this cycle much as they do the larger life cycle.

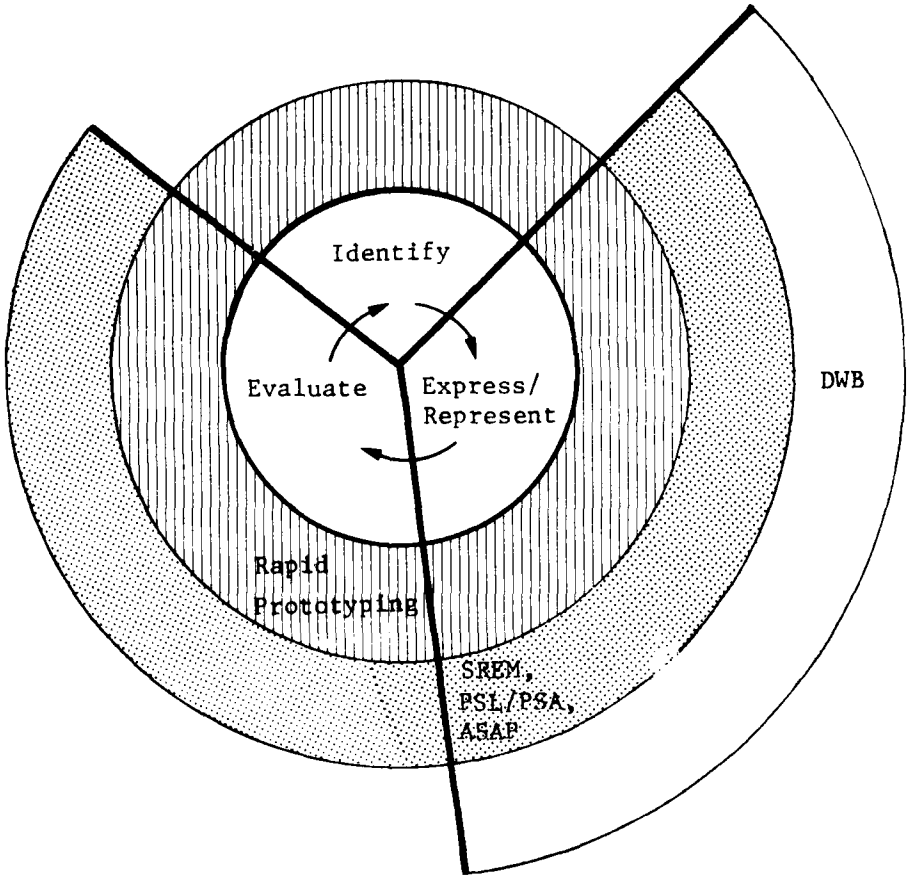


Figure 8.2-5
Adequate Requirements Definition, Software Tools, and Rapid Prototyping

Figure 8.2-6 pictures the interrelationships between prototyping, software tools and software development issues.

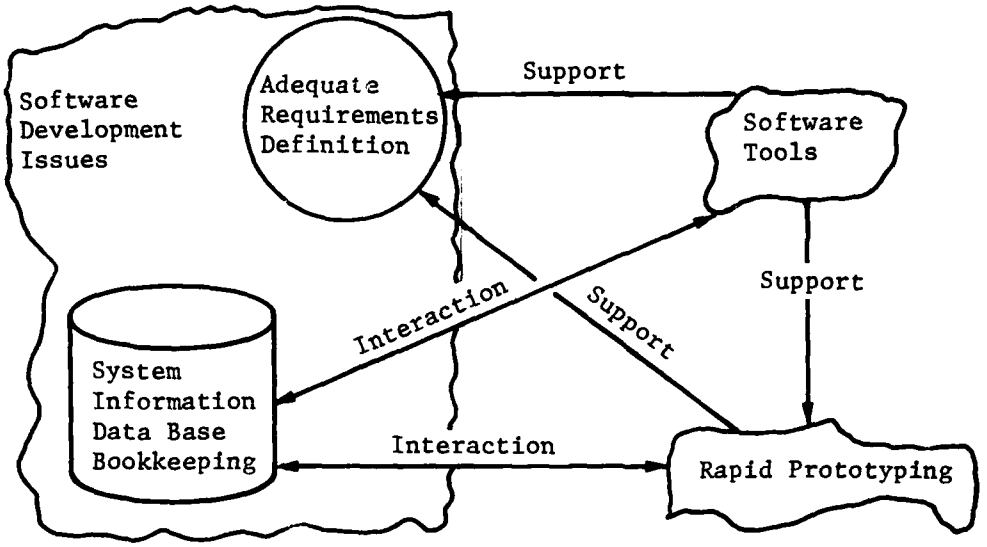


Figure 8.2-6
Interrelation of Software Development Issues, Software Tools, and Rapid Prototyping

8.3 RAPID PROTOTYPING SYSTEM (RPS) CONCEPT

8.3.1 Overview

The RPS will be a tool to prototype key functions of Air Force C^3I systems. The RPS must provide means to characterize the user-system interface aspects of a C^3I systems, its data base content and conceptual design, its structure at varying levels of detail, and its expected performance. The RPS must support documentation of the results as well as storage and retrieval of the prototypes themselves. At some future time, links to other systems and models may be required.

8.3.2 RPS System Elements

The RPS should be a hardware/software system designed to aid stabilizing and exploring C^3I systems' requirements and preliminary design concepts. It must aid the government in passing from a generalized Statement of Need (SON) understanding of C^3I systems to specific statement of work and/or to preliminary systems specifications for that C^3I system. It must be flexible and easy to use by a variety of players involved in the acquisition of command and control systems. Each possesses varying skill levels. There are three classes of users envisioned for the RPS: The prototype builder (who is also referred to as "software engineer"), the acquisition engineer, and the mission user. Each should be able to perform useful work with the RPS, both individually and in concert with other user types.

The RPS must emphasize the use of menu and icon accessed interfaces, multiple mode operation, and context sensitive system software. It must include high quality bit-mapped color terminals with sufficient display memory to allow the representation of motion as part of the definition of the user system interface. It should have high quality monochrome bit-mapped multi-window interfaces to support specification and functional analysis of proposed systems. The RPS can make use of the logic programming tool Prolog as a central part of its modeling and prototype specification abilities. Prolog can, in certain environments, run slowly. It would be wise, therefore, to include in the RPS hardware a dedicated system to process Prolog which features significant quantities of peripheral disk memory. The RPS can include a gate-way machine to link between Prolog processor, graphics processor, monochrome processor, and a VAX 11/780. The VAX can be used to support extensive system performance prediction modeling as well as to host supporting data bases. Additionally, there could be, within the RPS, several small individual user terminals on the order of an Apple Macintosh or IBM Personal Computer. These terminals could function as monochrome workstations processing a subset of the capabilities of the monochrome workstations specified herein.

8.3.3 User Types

The rapid prototyping system must be accessible by mission users, acquisition engineers, and prototype builders.

The mission user should be able to perform useful work with the RPS. He should access the system and perform interface modeling with an interface modeling component. The tools of this component should provide interactive, user friendly access. Software should be extremely context sensitive so as to provide as "modeless" an interface as is feasible for him. It should be the goal of the RPS to provide tools as easy to use as the LISA-DRAW^{R} paradigm developed by Apple Computers. The mission user should be able to draw screens and define required data base support in a transparent and friendly fashion. He should also be able to prototype knowledge based systems and to develop elements of a rule base as well as screens.

Menu based iconic and mouse accessed inputs should be the norm. Any data base functions necessary to support this should be transparent to the mission user, and access should be handled without his having to know any special query languages. The mission user further should access performance models. An operational model would allow the mission user to characterize the operational concept that he has in mind or has de facto prototyped with the interface modeler. This model could provide the mission user again with a menu and icon based friendly way to define the total set of procedures available to the analyst and his work station. The system should prompt the mission user for information necessary to define the run time requirements for an experiment. Certain inputs from the mission user may be appropriate initializations of a structured data base model. In particular, the mission user should find valuable contributions he can make to the data base model input templates. Complete exercise of the data base model component of the RPS would fall to the prototype builder.

The acquisition engineer must access and perform all the functions accessible by the mission user. Further, the acquisition engineer

should use the specification based prototyping capabilities of the RPS. This component should allow the acquisition engineer to easily define pictures and descriptions of structural subelements or functional subelements of the desired system. The RPS system must prompt the system engineer to make appropriate characterizations in test, structured language, and/or structured symbology in association with a functional element defined with this component. Further, he should be able to tie the elements of a connected graph type description to entry points within the interface modeler defined screens through the tooling component (in particular, the icon based executive). The acquisition engineer should use a logical modeling capability provided by the Performance Modeling component. This model should allow the acquisition engineer to capture logical descriptions of the system being prototyped in a manner so as to exercise them by scenarios developed by himself or mission users using the scenario generation component of the RPS. Once again, menu based or iconic based input means should be placed at his disposal.

The prototype builder is considered as the most technically sophisticated with respect to system and software design among the user types. The prototype builder must perform all the tool access discussed for the acquisition engineer and the mission user. Additionally, the prototype builder can make extensive use of the structured data base model so as to perform object-oriented analysis on the overall data structure required to support the system being prototyped. He should use the model through accessing Prolog based high level constructs that provide a friendly environment for this work. The prototype builder further could add to the descriptions or the detailed analysis of the system as captured in the connected graph descriptions. He should add PDL descriptions or pointers to PDL data bases to these descriptions. The prototype builder can make use of detailed ADP models so as to project performance of the defined hardware/software system. He should be able to supplement the interface modeling performed by the mission user with dynamic graphic elements (rotating globes and trajectory displays, for example). He should use the KBS prototyper to complement the work performed by the mission user so as to generate a first cut representation of KBS elements of the system being prototyped.

Significant functional aggregates involving screens, screen sequences, connected graph entities, performance prediction model exercising, and information content modeler queries and execution must be orchestrated by the prototype builder through use of the RPS executive. The prototype builder should make use of the icon accessed executive so as to quickly interconnect the capabilities provided by the different RPS components to make a dense functional prototype element. Generally, this would represent a C^3I component. Several Component System Elements (CSEs) may be joined by the prototype builder so as to aggregate a total C^3I prototype. The mission user or acquisition engineer should be able to make use of a top down specification capability to enforce a coherent specification of the system being prototyped. The prototype builder, on the other hand, would be able to prototype components "bottom-up."

The following scenario will demonstrate how each of these three user types may access the system.

8.3.4 Prototype Development

The RPS should provide the functions necessary to apply prototyping as an integral part of C^3I system development, with a well-defined strategy for its use. The strategy for the RPS in particular should be accomplished in the following phases:

1. Problem/Experiment Selection
2. Definition of Experiment
3. Specification of the Scenario
4. Development of the Prototype
5. Demonstration of Prototype
6. Evaluation

The strategy for accomplishing Item 4, the development of the prototype, would proceed according to the timeline pictured in Figure 8.3.4-1. The figure shows a relative scheduling of different types of prototyping which correspond to a top-down analysis of C^3I Support Environment.

First, high level modeling of the observables (i.e., the C^3I user system interface) would take place. Interface Modeling consists of creating mock-ups of the final system's screens, sequences of screens, and operator control events.

This is a means to make the system's gross observable states tangible and communicate an operational concept to the users of the system. In addition, database requirements, functional requirements, and performance requirements can be implied based on the screen sequencing and contents.

As interface modeling continues, database requirement issues accumulate. After a time, modeling of these database requirements can begin to resolve issues. Using the database modeling tools of the RPS, the consistency and adequacy of the database requirements can be determined. Also, relative cost for query and update of data can be determined.

After User System Interface and database modeling have progressed, an understanding of the functional requirements emerges. Modeling of the allocation of function to system components can then reveal problems in the functional requirements such as contradictions, deadlock situations, unacceptable resource contention and utilizaton, bottlenecks and other problems.

Early modeling will define single operator procedures well enough to begin to study the effect of several operators using finite resources. Modeling of Operator Procedures uses the single operator procedures and adds resource contention and functional performance estimates. Application of a scenario to the operator model will reveal contention and utilization problems resulting from the definition of the operator procedures.

8.4 SUMMARY

The lessons we have learned in the Denver Aerospace testbed have begun to define the role of rapid prototyping in the context of the maturing

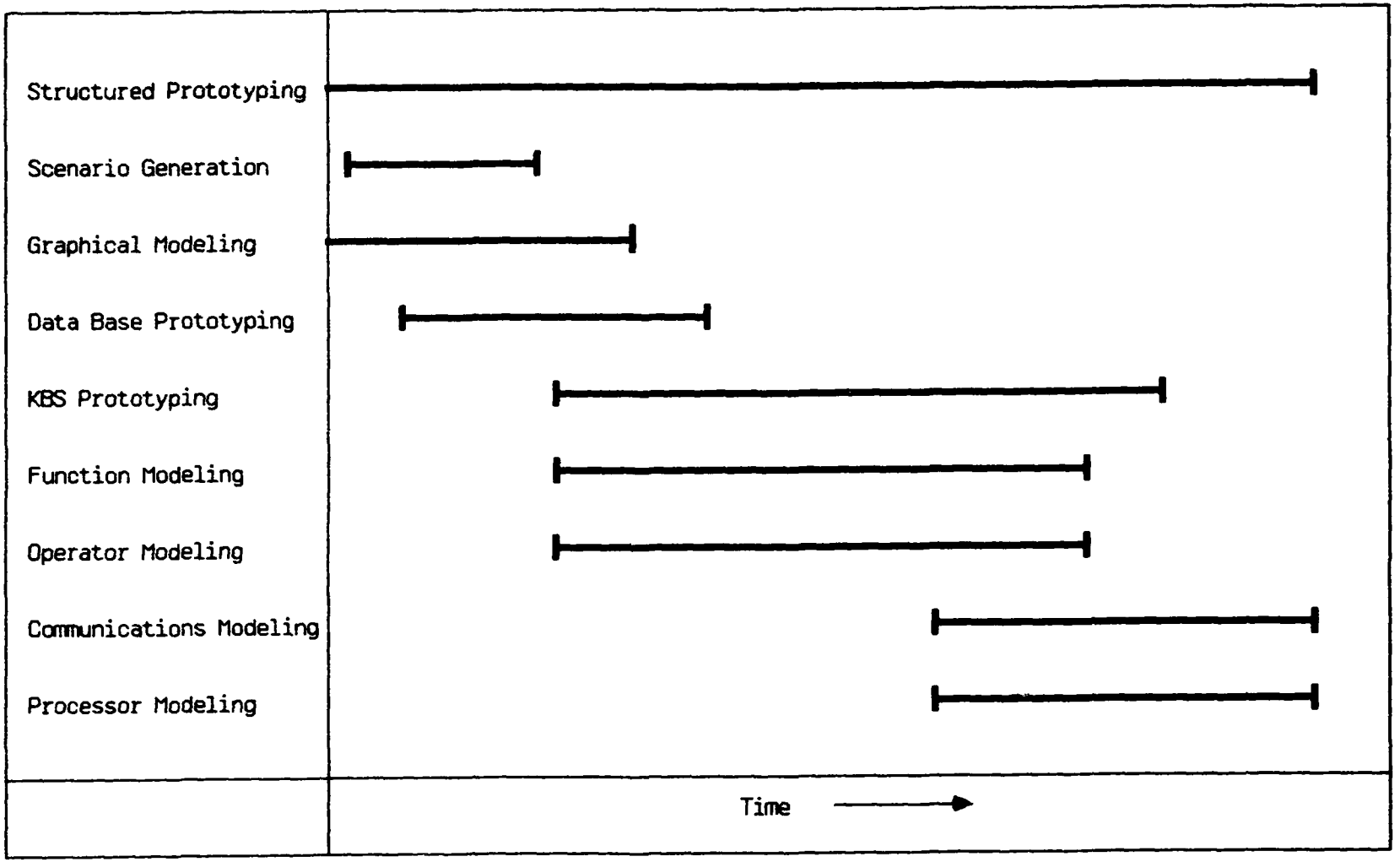


Figure 8.3.4-1
Top Down Prototyping Analysis Time Line.

software development discipline. Examination of this context revealed the interrelationship of rapid prototyping and other tools developed to meet the issues of successful development.

9.0 TESTBED ELEMENTS

In section 5.4 we discussed and rated the available prototyping approaches. To assemble and test a collection of instances of these approaches, we have to derive a representative set of capabilities. The capabilities selected must be available, and they must be feasible to implement and use within the constraints of our resources and staffing. With these constraints we have identified the following set of testbed components:

- an inventory of computer resources
- application software
- interface prototype
- a methodology
- scenario library

This list will no doubt evolve. These components are required to begin the process of defining and demonstrating a rapid prototyping capability.

9.1 Inventory of Application Software

The most visible part of the rapid prototyping environment will be the software. Software needed for the testbed falls into several categories:

- Model building tools,
- Software development tools,
- Data base tools,
- Existing databases.

Tools to build models are necessary to provide the core prototyping capability. This software must be reconfigurable and apply to one of the three aspects of concern in C^3I systems: Function, performance or user interface. Development tools must be included to perform three functions: The development of new models and simulations, the integration of prototyping activities with system development, and the modification of the rapid prototyping environment itself. Data base tools are necessary for developing databases to support models and manage project information. Existing databases store scenarios and descriptions of logical models.

9.2 Computer Resources

Adequacy is measured against the needs of software to be hosted, the state of computer system architecture, and the needs of the ultimate end-users of the prototyping environment. A major trend in architecture toward distributed networks of computers and peripheral devices deserves some mention. The trend is driven by a number of factors. Advances in the development and production of hardware has brought down cost, allowed for increased specialization of hardware and concentrated processing power in smaller, cheaper packages. At the same time users need and desire the sharing of information and resources. Tying these two factors together is a growing legacy of software and industry standards which support network compatibility across a wide range of machines. Hardware and software advances are making users' demands technically feasible and affordable. The centerpiece of this trend is the local area network (LAN). In a LAN, several computers and peripherals may be tied together and accessed by many users operating from

intelligent workstations. This scheme allows users to exploit a variety of machines in a manner efficient from the user's perspective and the system designer's perspective. One machine could never achieve the generality, response, and efficiency of LAN interconnected special purpose processors. These advantages will be as apparent to the end user of a rapid prototyping environment as any other user.

Certain specific needs of the rapid prototype user population emphasize the need for a distributed network of workstations. We can call these needs multiplicity, segmentation, and heterogeneity. Multiplicity is the need for many copies of the environment. This can be achieved by simply plugging as many workstations as necessary into a LAN. Segmentation refers to the need to segregate groups of users working on separate projects or phases of projects. This can be achieved by isolating groups of workstations and resources in the network physically and/or logically. Heterogeneity means that these same capabilities may be implemented by different machines in different instances with the same interface provided. On a LAN, the intelligent workstation would serve as a logical buffer, hiding implementation details of a given capability from the user, providing a common interface for all users regardless of their implementation. In another sense, heterogeneity refers to the users. Users of the prototyping environment will be differentiated by technical expertise and motivation for using the tool. In response to this, workstations can be customized to meet the different needs. All of these concepts must be incorporated in the testbed's computer resources. This requires a distributed computer system architecture accessed by an intelligent workstation.

9.3 Interface Prototype

We require a fluent method of accessing the testbed software. This access is required both for informal testing of the environment as it is being assembled and for customer interaction for requirement and design validation. The best way to provide this access is as a prototype interface. At this stage the prototype must be reconfigurable and provide a general enough framework to house the known components of the testbed as well as those components not yet known.

9.4 Methodology Requirements

A methodology is required to bind the components together into a usable package and to integrate these capabilities with the system development life cycle. An ideal automated design support system (for computer aided software engineering/system design) would be at the same time general in application, aid the design process, support test and integration, and cut across design disciplines. It would also complement code generation and programming environment tools.

A methodology for the RPS will consist of a process for using its tool components. This process can be seen as a plan for sequential use of the tools as shown in Figure 9.4-1. Of course, the sequence of tool use can vary depending on the prototyping problem under consideration. Figure 9.4-2 shows that there can be several paths through the tools, each one valid.

To support the expression and validation of system requirements, the RPS methodology would emphasize structure prototyping, interface prototyping,

and performance prediction (especially gross data flow analysis via COMS and GOM). As system requirements stabilize, the expression and validation of software requirements would be important. Although derivation of software requirements from system requirements is possible, it becomes more important to understand the proposed system in depth rather than formally deriving specifications. The RPS users would move to more detailed structural analyses using a diagram constructor and the FAM. They would extend the functionality of the interface prototype through characterizing the dynamic graphics portions. Any knowledge based system components of the C^3I system under consideration would be investigated at this time. Its data base system would be characterized to some level of detail which could later be extended. Later still, as preliminary or conceptual design becomes important, the RPS users would use the GPM to support identifcation and flowdown of performance requirements.

Further, all previous models of the system would be detailed and extended.

The general plan for prototyping is "analyze-model-demonstrate" iterated many times. Analysis should be unconstrained by the prototyping tool. Modeling must allow several types of representations of the system to exist simultaneously. The human development process is a group activity. Consequently, the RPS must allow several users simultaneous access and provide support tools that aid each separate view of the system as shown in Figure 9.4-3.

9.5 Scenario Library

Testbed components, together with a methodology, comprise a workable prototype environment. To drive that environment requires operational scenarios. The scenarios must be realistic, and we must have measures for the performance of the environment. To then measure the performance of the environment in use, we need a record of the results using conventional developments methods. In sum, a history of C^3 system development is required. This history may take the form of written documentation, computer storage media, and people involved in the projects. Figure 9.5-1 illustrates how scenarios are derived from C^3I systems development history to drive measurable results from the testbed.

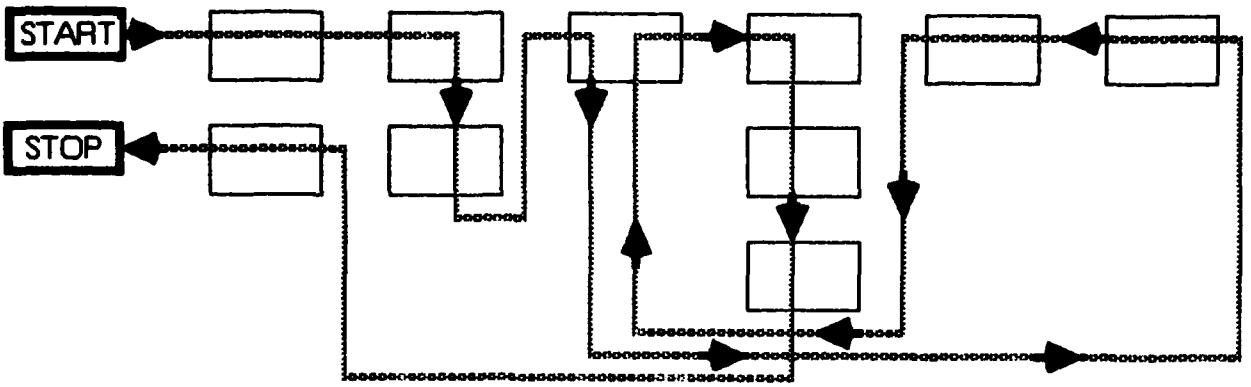


Figure 9.4–1 Definition Of Method–a Plan For Tool Use Method Is Tool Sequence.

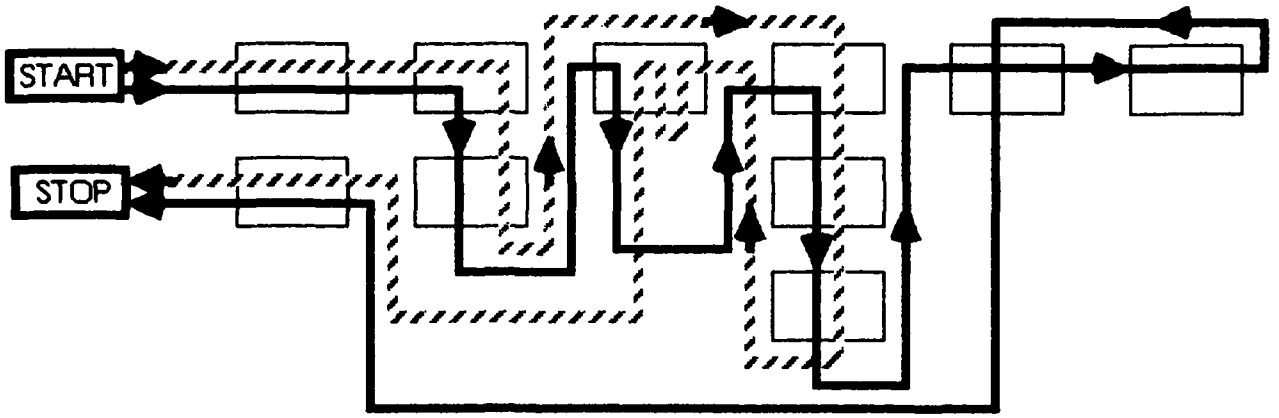


Figure 9.4–2 Our Tools Can Be Used With Different Methods.

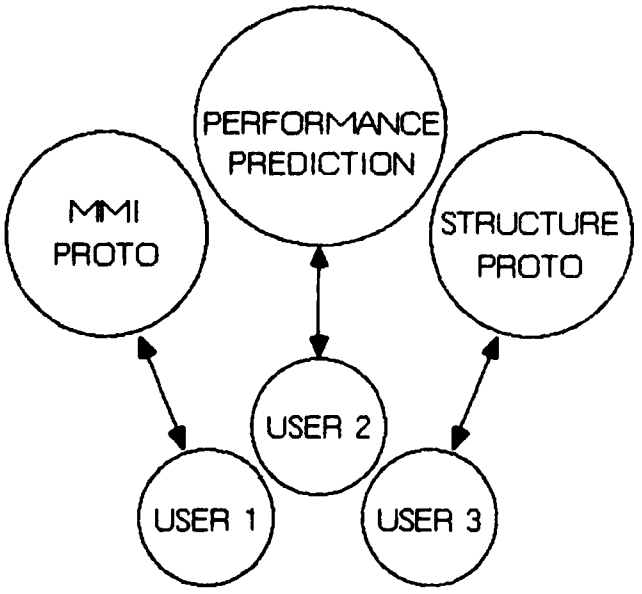


Figure 9.4–3 Tools Can Be Used Simultaneously And Emphasized Or Deemphasized.

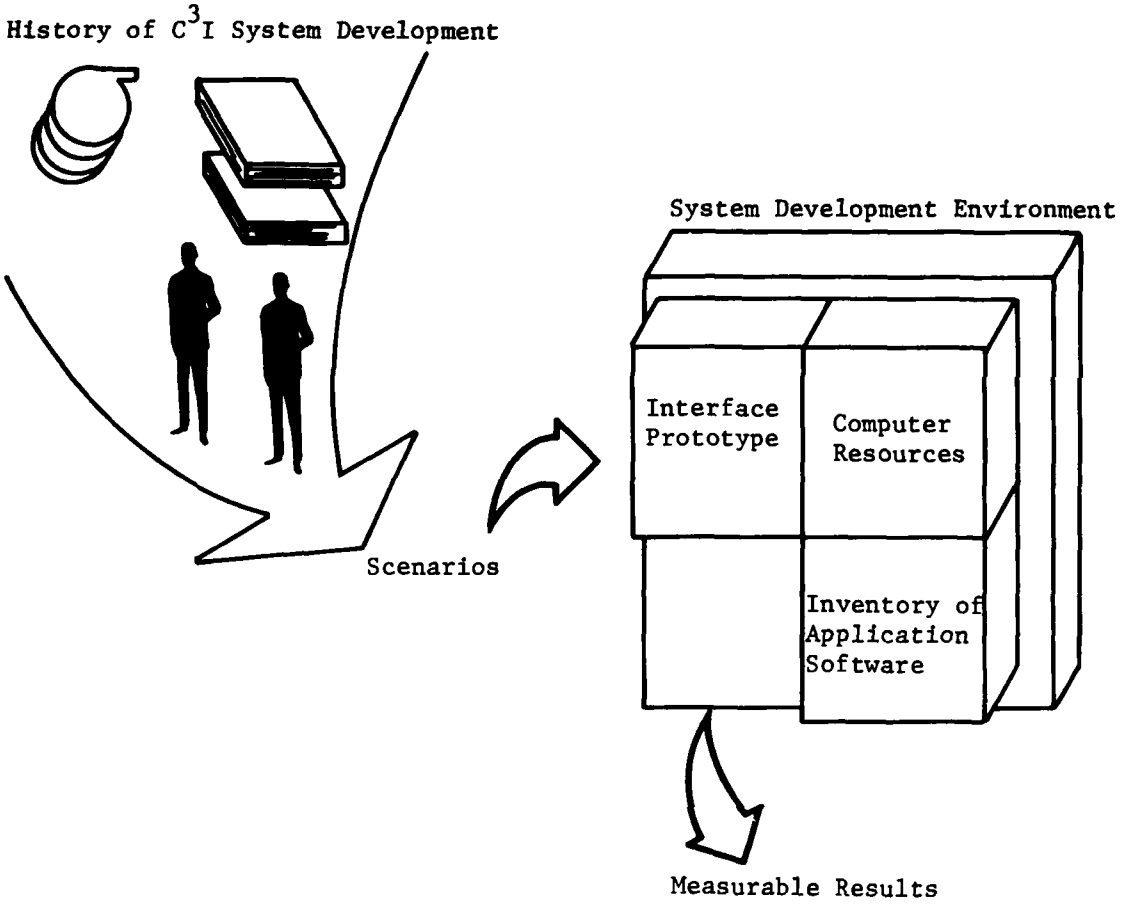


Figure 9.5-1
The history of C^3I system development produces scenarios to drive the test bed.

10.0 CURRENT TESTBED

10.1 Inventory of Application Software

We currently have representative software for each of the required categories: Existing model building tools, software development tools, data base tools and existing data bases. In Figure 10.1-1 the software we have available is pictured.

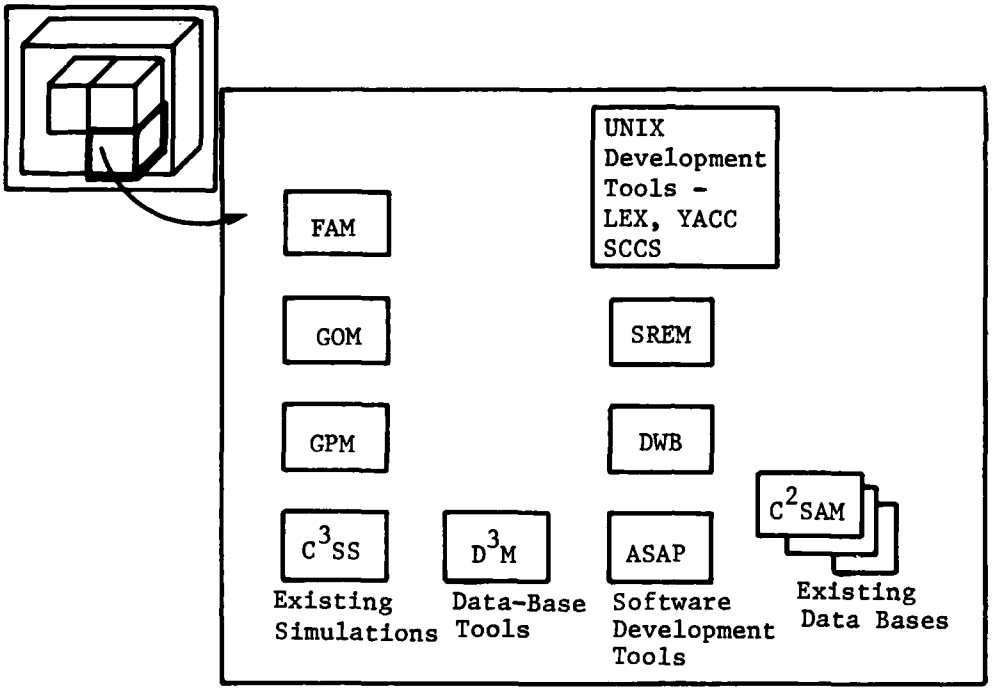


Figure 10.1-1 Inventory of Application Software

Existing Model Building Tools

We have focused closely on certain of these simulation languages that appear to be the most usable candidates. Among them are these three simulations (which we have acquired and studied): The Functional Allocation Model (FAM), the General Operator Model (GOM), and the General Processor Model (GPM). In addition, we have available the extensive software in Martin Marietta's C^3 System laboratory.

FAM is a high level functional model. It characterizes a system by using stimuli, threads, functions and resources as shown in Figure 10.1-2. It drives the modeled system with scenarios and events. These parts are related in the following ways: Scenarios are lists of events; events give rise to certain stimuli; stimuli invoke processing threads; threads are made up of sequences of functions; functions contend with other functions for resources. The model is instantiated by analyzing operational threads, available resources, etc., of the system; representing that analysis in the model's data base; and then constructing scenarios, also stored in the database, to drive simulations. Measurements, taken during runs, on resource contention and

other performance aspects of the system are a means of evaluating the functional allocations made in the system.

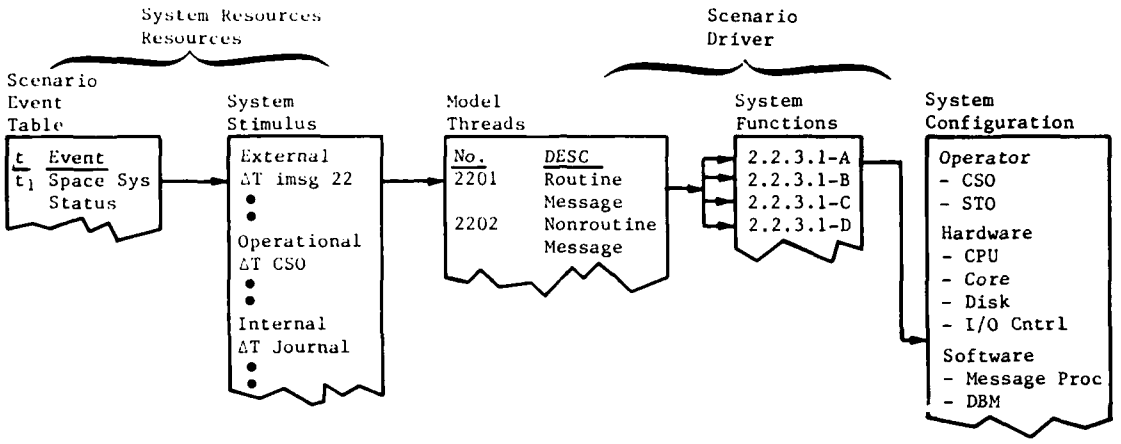


Figure 10.1-2
FAM uses a series of data bases to describe the logical structure of a system.

GOM models a system in much the same way as FAM. To set up the model, the system is described by specifying the number of operators, what tasks they perform, how long it takes them to do tasks, what resources they require for tasks, and what resources are available. The model is driven by a scenario of events that stimulates operators to perform certain tasks. From a simulation run, statistics on time in system and resource contention are collected. Using empirical data describing the way operators use different equipment, the modeler can run simulations to compare alternative operational systems.

GPM models in detail a computer system. Generally, it is a discrete event model of the processors, memories, certain operating system functions and the I/O of a computer system. Scenarios consist of lists of processing tasks to be submitted to the hypothetical system. Scenarios are stored in a database.

Each of these three was developed using FORTRAN within the SLAM modeling framework. As a result, they are able to be combined. FAM or GOM could be combined with GPM to move from a highly abstract model to a more detailed model. This would essentially be a nesting process, where the system is first modeled using FAM or GOM alone, and then GPM is substituted for the simple characterizations of processing resources to achieve a greater level of detail in the simulation.

The C^3 laboratory system software runs the suite of devices in the C^3 lab. This includes a large scale graphic display and graphic workstations. Like the other models, the C^3 models are configured and driven from a database.

Sortware Development Tools

We have been interested in software development tools for some time. We used, evaluated, and developed new tools. In pursuit of this interest, several packages have been evaluated. PSL/PSA (Problem Statement Language, Problem Statement Analyzer) was studied and used on projects in the company. Denver Aerospace also conducted a lengthy evaluation of TRW's SREM (Software Requirements Engineering Methodology) for the Air Force (RADC/CO). Tools such as ASAP and DWB have been developed by Martin Marietta. ASAP (Automated Structured Analysis Package) is a design aid that has been used and is being used. It is based on Yourdon data flow structured analysis techniques. The tool can be used in storing and managing design information and provides completeness and consistency measures on data flow aspects of a system being designed. This tool is applicable to the system development life cycle from early requirements definition through implementation. It can be used to manage project information, including such concerns as project organization and schedule, as well as requirements, decisions, detailed design, and even source code.

DWB features flexibility. The structure of the project data base is flexible; the specification languages used can be tailored to suit unique needs; and hooks are provided into the host operating system to interface DWB with other tools. Unlike ASAP, DWB does not support a specific design technique but rather focuses on the management and expression of project information in such a way that users are facilitated in using any development techniques they choose. For instance, ASAP might be used along side DWB in the design phase of a project.

Of these tools we are most interested in SREM, ASAP and DWB. We have all three available to us and have studied them. We have hands-on experience with ASAP and DWB. Right now we are in the process of setting up a DWB data base to support this project. Appendix A1 presents an example of a description of our methodology, described using a SREM-like syntax, stored in DWB. ASAP was used in the development of our interface prototype, and we expect to use it in our design efforts for other parts of this project.

Some other tools we have available are parsing code generators that can be used in the rapid implementation of application front ends--in our case, prototype applications. On the UNIX operating system we have available LEX and YACC. LEX is a tool that generates source code for Lexical Analyzers from descriptive source code of its own. This not only allows the rapid generation of a Lexical analyzer, but also promotes easy modifiability. YACC (Yet Another Compiler Compiler) generates source code for parsers. It can be easily teamed with LEX generated code. Lang Pak is another parser generator available to any system with a FORTRAN compiler. It has a descriptive language in which to describe a grammar. Then the grammar description is translated into Fortran source for compilation and use. In addition it provides an interactive environment for developing a grammar.

Data Base Tools

We currently have in the testbed a data base generating tool based on the Codasyl data base standard with a relational access language. Under this standard a data base scheme is constructed from a description of the data

types to be stored and the relationships between the data types included in the data base. The scheme is a translation of the data descriptions into an access structure which can be used to populate and query the data base. The particular tool we have is called D3M for Domain Distributed Data Management. It is hosted on the Apollo DN300 workstation, which will be described later. In addition to generating a scheme for a data base, D3M also will generate subschemes for restricted access and has a built-in interface to the high level languages Pascal, FORTRAN and C.

Existing Data Bases

A notable data base is the C2SAM, or Command and Control System Analysis Model. The information in C2SAM and the underlying model for organizing that information comprise a generic logical model of any command and control structure.

Five groups of elements make up the model;

- Missions,
- Levels,
- Hierarchical Breakdown of Generic Tasks,
- Information Units,
- Organizations.

All command and control is exercised in the support of some mission. Each level of command and control performs certain tasks to accomplish a mission. Different levels may perform the same tasks. Tasks are generic activities that must be carried out by any C2 system in pursuit of its mission. Certain tasks require or produce information units. Organizations perform certain tasks.

The C2SAM is Air Force specific. The organizations themselves reflect current structure in the Air Force. Some, although not all, of the information products are standardized Air Force products. The missions used in the model are all air missions. This apparent lack of generality reflects the current degree of instantiation of the data base. C2SAM currently represents the TAC, MAC and USAFE (CENTCOM) organizations. To extend the model, the data base must be populated with specific information relevant to other organizations. Figure 10.1-3 is a high-level diagram of the generic information exchange model as it exists in the current instantiation of C^3SAM. The following paragraphs expand somewhat on the fundamental elements of the model.

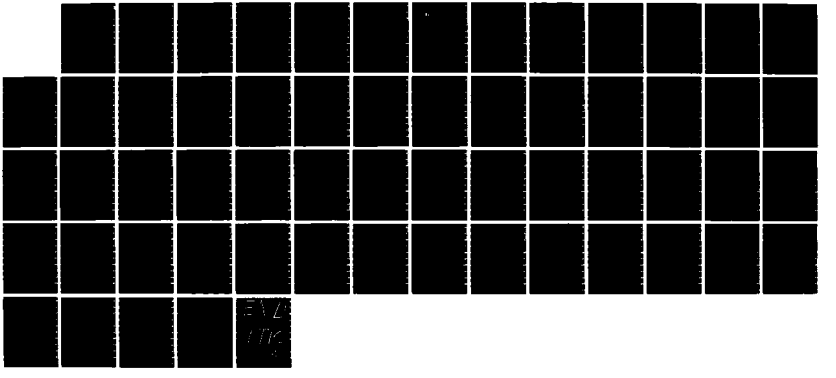
Five missions are enumerated;

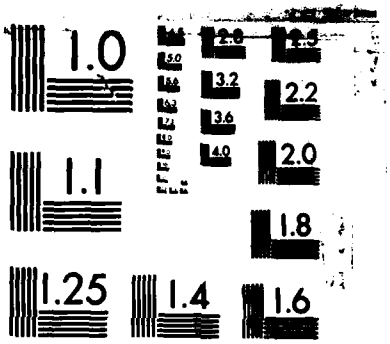
- Offensive Counterair,
- Defensive Counterair,
- Close Air Support,
- Interdiction,
- Reconnaissance.

Certain tasks are associated with each mission. The missions may also be combined, and some tasks support all missions.

AD-A167 423 C3I RAPID PROTOTYPE INVESTIGATION(U) MARTIN MARIETTA 2/2
AEROSPACE DENVER CO P C DALEY JAN 86 MRC-85-616
RADC-TR-85-216 F30602-83-C-0067

UNCLASSIFIED F/G 17/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

Four levels describe the command and control structure;

- Theater,
- Component,
- Force,
- Unit.

Theater responsibilities are very broad, component responsibilities less so, and so on. Each lower level generally deals with more implementation detail. However, exceptions may crop up where higher levels of command and control are involved closely in planned execution.

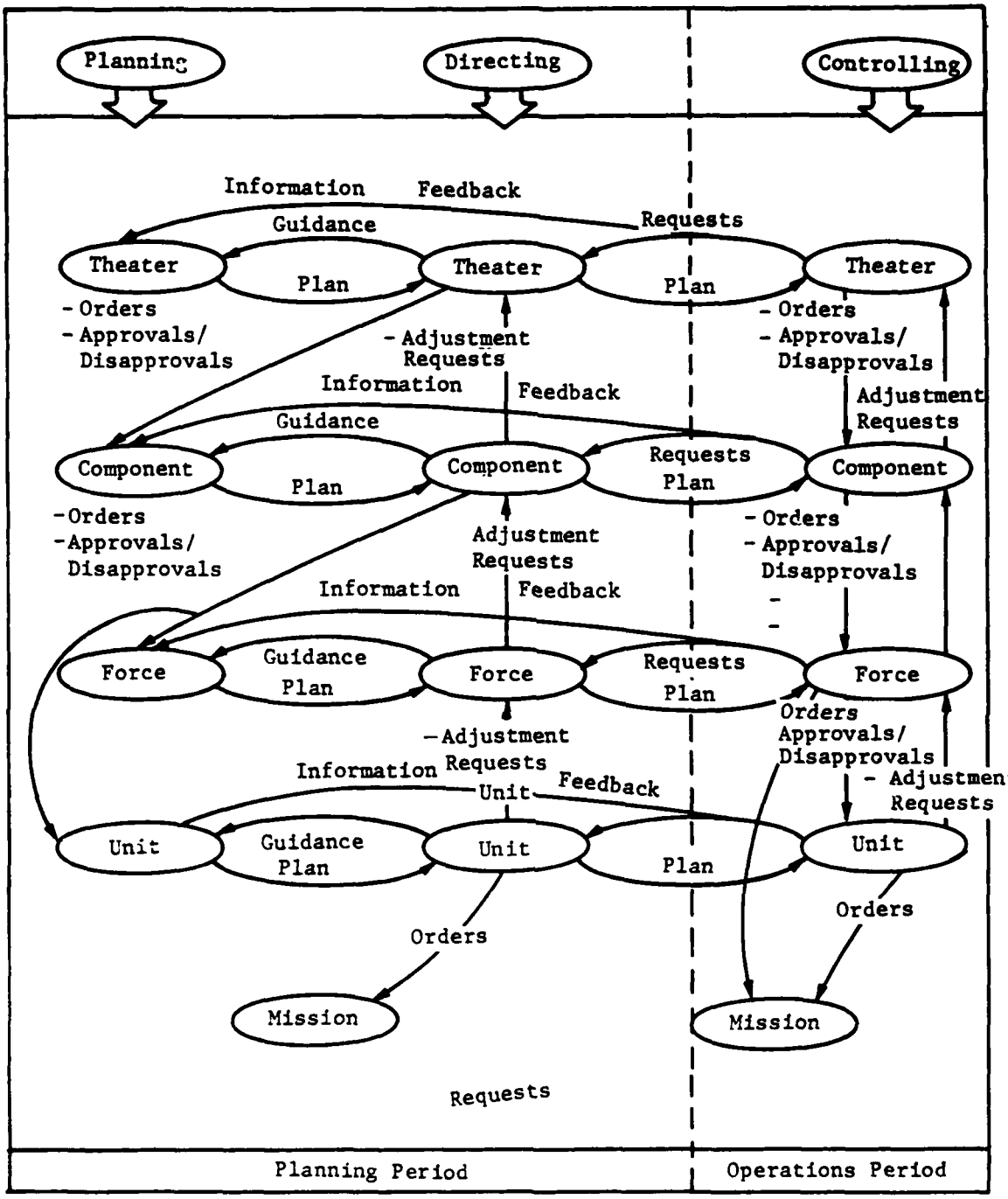


Figure 10.1-3
Generic Information Exchange for Tactical Air Missions

The hierarchy of tasks includes five levels of detail as follows;

- Function,
- Subfunction,
- Activity,
- Subactivity,
- Job.

Planning, directing, controlling, and executing make up the list of functions. Lower level tasks quickly proliferate. Subfunctions number 65, activities 221, sub activities 831 and jobs 1575. The lowest levels, subactivity and job, are directly associated with the transfer of information. Organizations are structured hierarchically, also. Each organization is related to particular tasks. Tasks may be shared by organizations. Information products are categorized as formal and informal. They describe groups of items of information that are necessary for the execution of tasks. A user queries the data base to use the model. By querying in an investigative fashion, a general functional model of a prospective C2 system can be constructed.

10.2 Computer Resources

Hosting the software inventory and satisfying the other constraints on computer resources was accomplished by using the Martin Marietta Central Software Laboratories (CSL) and an Apollo Domain System.

The Central Software Engineering Facility (CSEF) consists of a network of six mini and super-mini computers including Vax 11/780, PDP 11/70, IDM 350, and IBM 4341. A wide variety of peripherals connects to this network, including printers, plotters, graphic terminals, graphic terminal hardcopy devices, and a microfiche peripheral. We used two VAX 11/780s, one running VMS operating system and one running UNIX. Also, we have used the IDM 350 machine which is connected to the VMS VAX and supports Designer's Work Bench. CSEF hosts all of the software except D3M and the interface prototype software.

We chose the Apollo DN300 computational node as our workstation for several reasons. Specifically, we were impressed by the processing power, the operating system and the network capabilities of the node.

The node's processing is done by two Motorola MC68010 microprocessors. These microprocessors are state-of-the-art 16-bit processors. They have built-in memory management hardware. The instruction set for the MC68010 directly supports structured high-level languages and current operating systems.

Apollo's Aegis operating system successfully exploits the hardware. Aegis interfaces with the user through a high-resolution bit-mapped graphic monitor. The interface features a multiwindow display similar to systems such as Xerox Smalltalk, the LM-1 or Apple's Lisa. These other systems, however, are isolated to comparatively narrow applications. Aegis uses the innovative interface to facilitate conventional application software development. This approach brings along the advantages of the new concepts while not isolating the user from mainstream production methods.

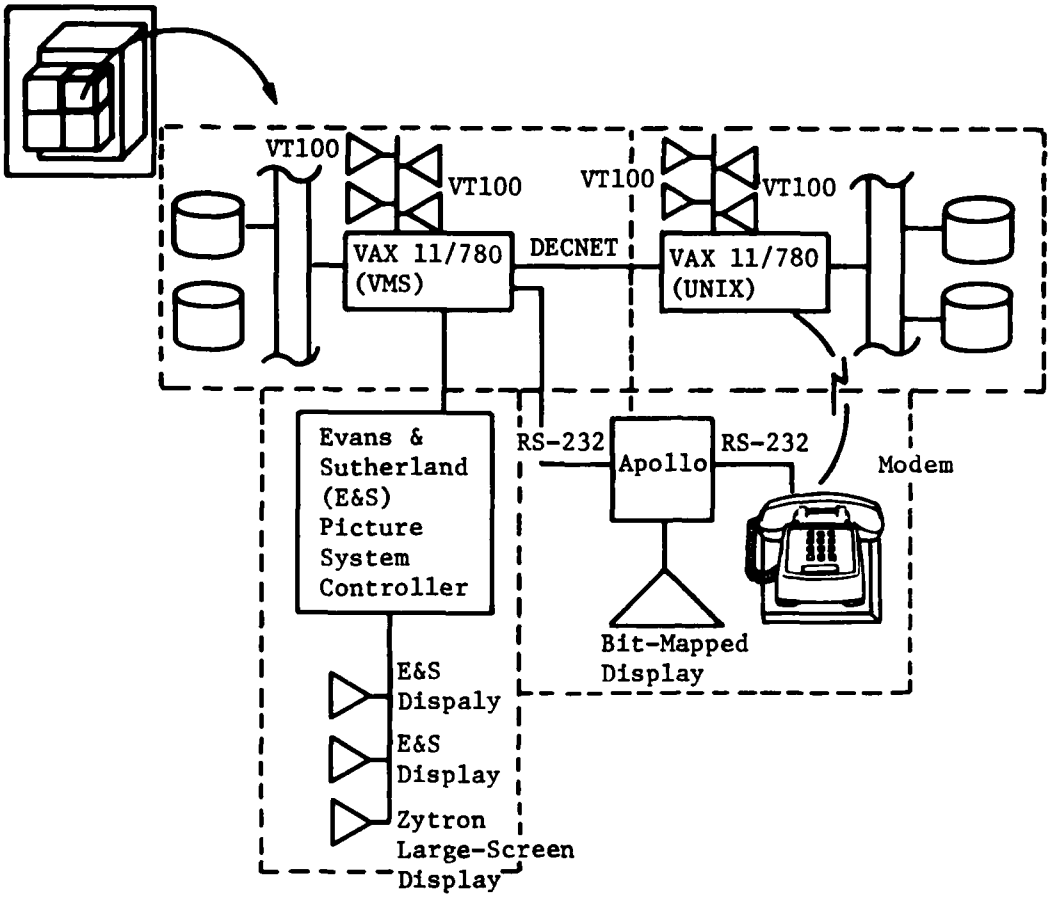
Cognizant of the trend to distributed systems, the Apollo designers have built network capabilities into the workstation. The node can be tied into a LAN in three ways. It may be plugged directly into an Apollo Domain Network; it can be connected to computer hosts or modems via two RS232 ports; or it can be gated onto an industry standard network, such as Ethernet, by use of a peripheral server node.

Apollo Domain Network is restricted to Apollo products, but this LAN interfaces through the Apollo DSP80 gateway node to other LANs. The Domain network architecture is based on a ring topology utilizing a token passing protocol. This architecture is extremely fast--rated at 10 Mbytes/sec. In addition, all of the products offered for the network are completely compatible. The operating software in all the available nodes includes network administration functions. All of the features in combination form a virtual network, one in which any user, anywhere on the Domain network, may use any resource on the network as if the user were connected directly to that resource. No appreciable response delay occurs as a result of network communication, and the drawbacks of Ethernet load management are avoided. Resources available within this virtual network run the gamut of commonly-used engineering tools. Special packages are available for CAD/CAM application, database system, scientific engineering, large secondary storage, specialized peripheral devices, and more. The Domain network is sufficient for a great many applications in and of itself.

The network provides a gateway to other networks. This gateway is in the DSP80 peripheral server node. A DSP80 node was included in the network to interface the network with special peripheral devices or other networks. In particular, Apollo offers software for the DSP80 to directly link Domain network to an Ethernet or HYPERchannel. This capability allows the user to enjoy the extremely high performance Domain network and at the same time does not isolate the network from standards which are developing in industry.

Directly linking the DN300 to a host or modem using the Serial I/O lines allows it to be used in a terminal emulation mode. Software included in the operating system emulates an ANSI standard terminal and can be configured for various operating system conventions. The software also transfers files to and from a host. These lines may also be tied to a peripheral device such as a printer.

We used the Serial I/O ports to connect the workstation to the CSEF network. One port was tied directly to a VAX host running under VMS. The other port was connected to a modem. Figure 10.2-1 diagrams the demonstration configuration. This configuration is a minimal example of the LAN architecture we have described. The LAN in this case is the DECNET of six computers in CSEF. With the Apollo in terminal emulation mode we are directly connected into the CSEF network. From our workstation we ran software on two separate VAX machines, one running VMS, the other running Unix. Software hosted by the workstation provided a framework within which to control simulation running on one of the VAXs.



Current C^3 Laboratory Displays

Figure 10.2-1 Minimal LAN Configuration

10.3 Interface Prototype

Our minimal network/workstation configuration brings together the individual capabilities to define and demonstrate a rapid prototyping environment. To begin defining the environment in terms of the user interface, we developed prototype software on the Apollo. The prototype was designed to represent a framework into which the prototyping tools could be placed. A user would use this framework for guidance and access to the tools. It was also designed to be flexible and reconfigurable. Figure 10.3-1 is a diagram of the software running the prototype. The drawing is partitioned into the user's view, Apollo hosted processes, and VAX hosted processes.

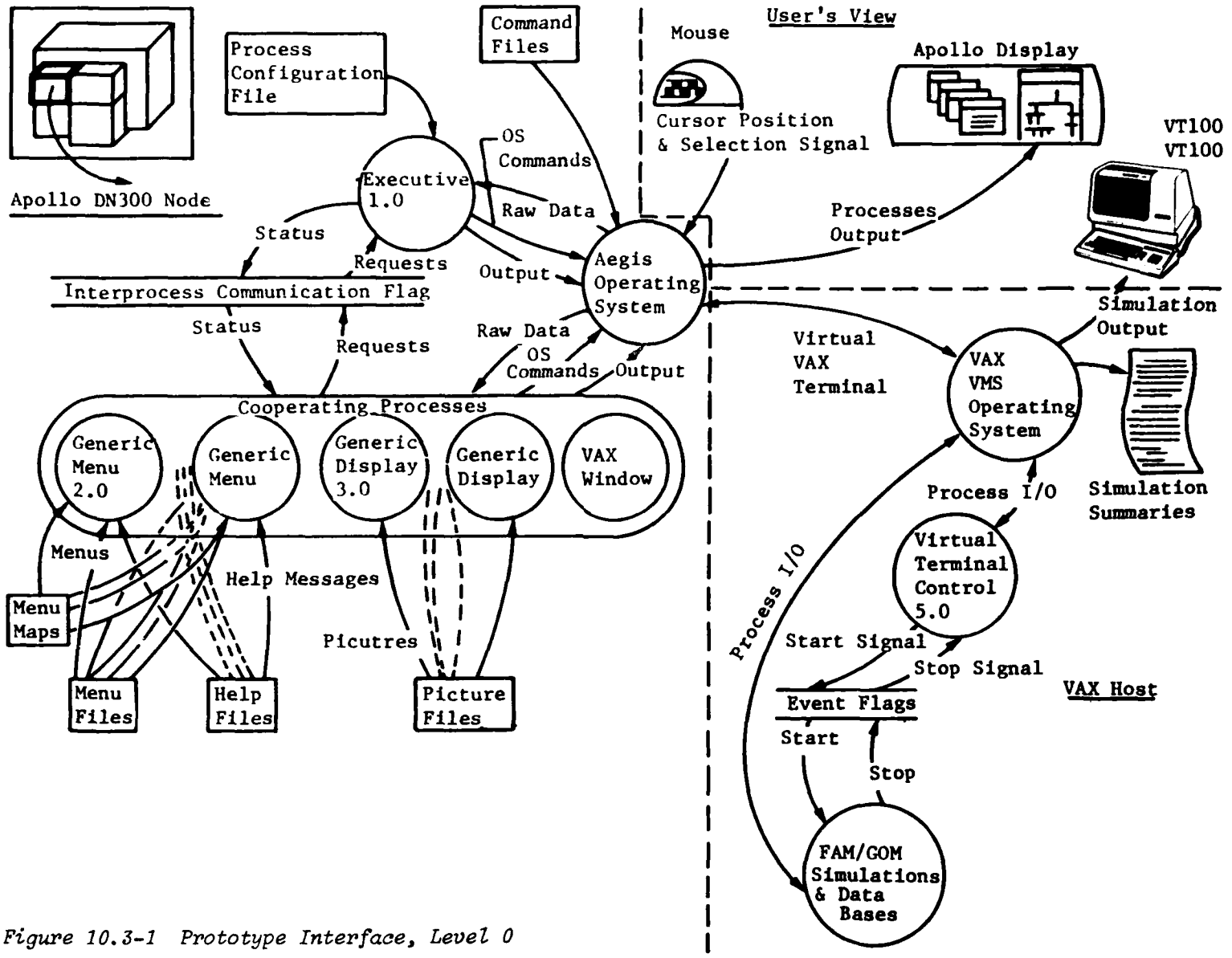


Figure 10.3-1 Prototype Interface, Level 0

Essentially, the prototype consists of an executive process and several cooperating processes configured by several files. The executive process functions quite simply as depicted in Figure 10.3-2. It begins execution by reading a configuration file into a table. Then it maps into its address space a small area, called an Interprocess Communication (IPC) flag, used for communication with the other processes. After initialization the executive simply monitors the interprocess communication flag for events. Upon the detection, "flagging" of an event, the executive examines the contents of the communication space and takes appropriate action. If the flag is set to a positive integer, then this integer is used as an index into the configuration table. Information indexed by the flag tells the executive what process to initiate. In the case of a negative flag, the executive terminates a process. While the executive monitors events it also keeps track of what processes are already active and how many processes are remaining. The flag is set to reflect these conditions. That way, if a request is made to begin a process that is already running, then the requesting process can determine that fact from the flag after it has made the request. Similarly, if so many processes are running that no more may do so, then the flag is set to so indicate.

The executive is the means for the prototype's other processes to cooperate. A process may request the initiation of another process or inform the executive of its own termination. After each request, the process then waits for an acknowledgment from the executive indicating the status of the request.

This structure can be used to implement a menu system. Figure 10.3-3 is a diagram of the generic menu process used. The menus in this system presented the user with a choice of functions in the prototyping environment. Selection from the menus was made by simply positioning the cursor over text on the screen and clicking a button. Positioning was done using a "mouse" pointing device. The user selects activities from a menu, activities which might be from another menu, or one of several other types of processes. Included in the demonstration were processes for configuring and running the FAM, displaying graphics, and editing graphics.

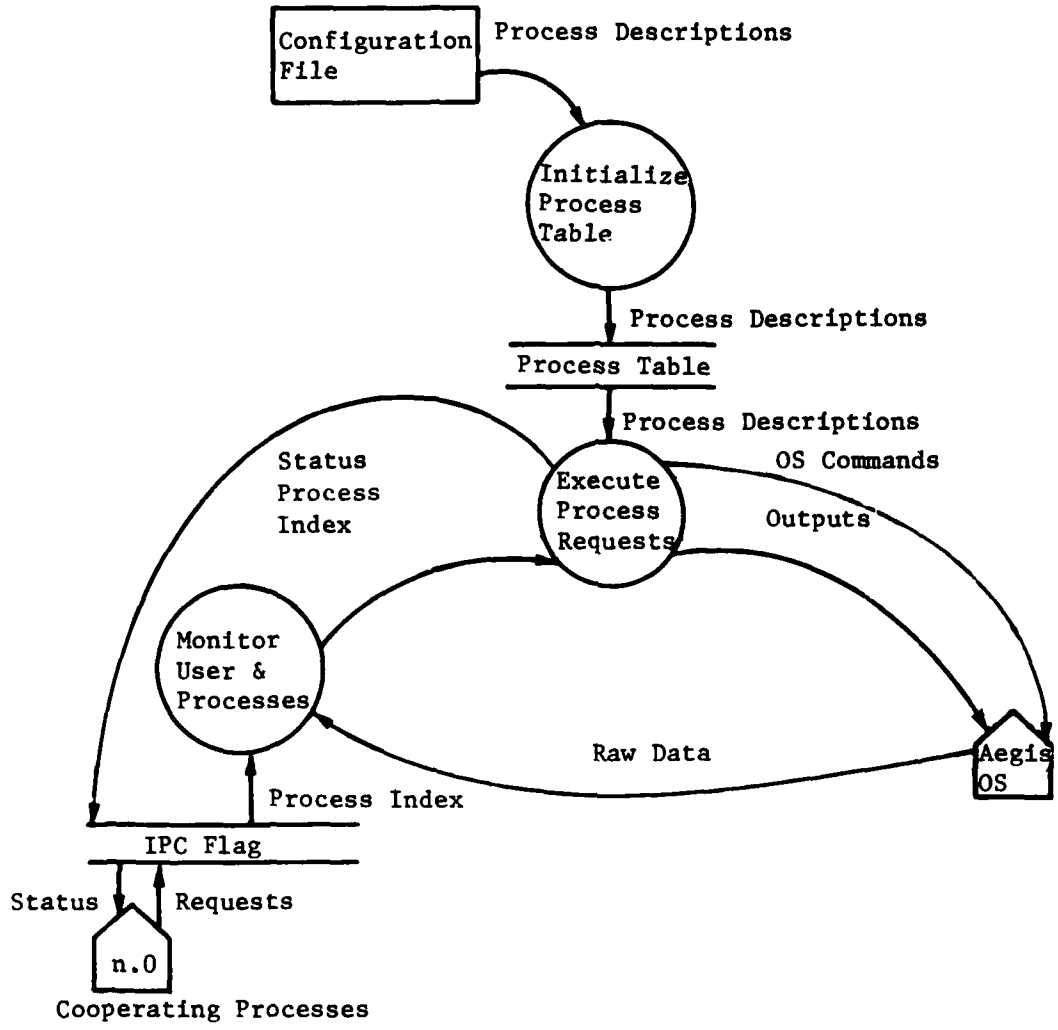


Figure 10.3-2 Prototype Interface Executive

In the menu system itself, we provided two means of access to the user: a hierarchical method and a direct method. The hierarchical method begins by listing the highest level activities first. Selection of a top-level activity then results in the presentation of a menu of lower-level activities and so on, until the user has selected a core prototyping activity. This hierarchical method of access was designed to guide the user through the methodology. As users become more familiar with a system, the hierarchical method becomes cumbersome rather than helpful. Therefore, we also provided a direct method of access. The direct method consists simply of an exhaustive menu from which the user may select any of the processes available.

10.4 Methodology

The methodology must define when to prototype, what to prototype, the means to use, and what to do with the results. The methodology we have developed thus far answers all of these questions to some level of detail, but

it focuses most clearly on the question of when to prototype. This question must be answered because it addresses the most deeply rooted problems in the development of software for C^3I systems.

The upper level of Figure 10.4-1 helps to explain why the 'when' of prototyping so critically concerns the development of software. The diagram depicts three phases proceeding in time from left to right. The first phase activities identify, express and validate requirements for the system. From the system requirements, Phase 2 identifies, expresses and validates the resulting requirements for software in the system. Phase 3 activities result in a software design.

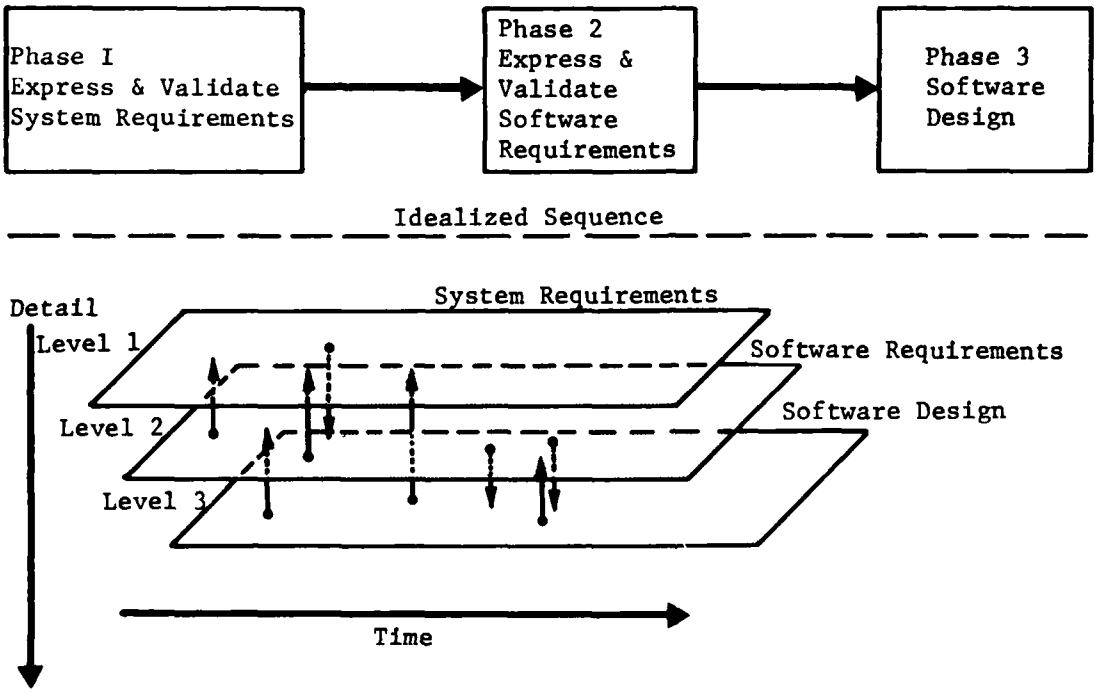


Figure 10.4-1 Realistic Model of Activities

This sequence of activity reveals the dependence of design on software requirements and software requirement on system requirements, which is the hallmark of modern software engineering. Each succeeding phase is dependent, for its own validation, on the validation of the previous phase's results. Prototyping facilitates validation. To effectively build good software, prototyping should be applied in the system requirement process. The top half of Figure 10.4-1 may be considered an idealized sequence of activities, idealized because the flow of activities in each phase is not truly sequential. The dependencies are sequential, the processes are not.

The lower half of Figure 10.4-1 illustrates a more realistic model of the development process. The phases are now pictured as levels or planes of activities. The position of the left edge in relation to the timeline indicates the sequence of initiation of the phases. As you can see in the diagram, the durations of the phases overlap. The analogy says that

activities in all of the phases can take place at the same time, some in parallel, others asynchronously. Activity proceeds in an opportunistic fashion as the necessary inputs become available and management constraints are satisfied.

As a result, requirements are validated in an ongoing way at all levels. A validated baseline of requirements, however, must be arrived at as quickly as possible. We consider it critical in our development of a methodology to facilitate validation in a manner that recognizes the overlapping in time of essential activities and, at the same time, the need to quickly formulate a baseline.

The three phases pictured in Figure 10.4-1 partition our methodology. Figures 10.4-2 through 10.4-11 show detailed activities within the phases. Phase 1, Express and Validate System Requirements, is detailed in Figures 10.4-2 through 10.4-6; Phase 2, Express and Validate Software Requirements, in Figures 10.4-7 through 10.4-10; Phase 3, Software Design, in Figure 10.4-11. In the diagrams, activities are represented by circles, products of activities by rectangles, and on-line storage by cylinders. Arrows indicate the flow of information in the methodology.

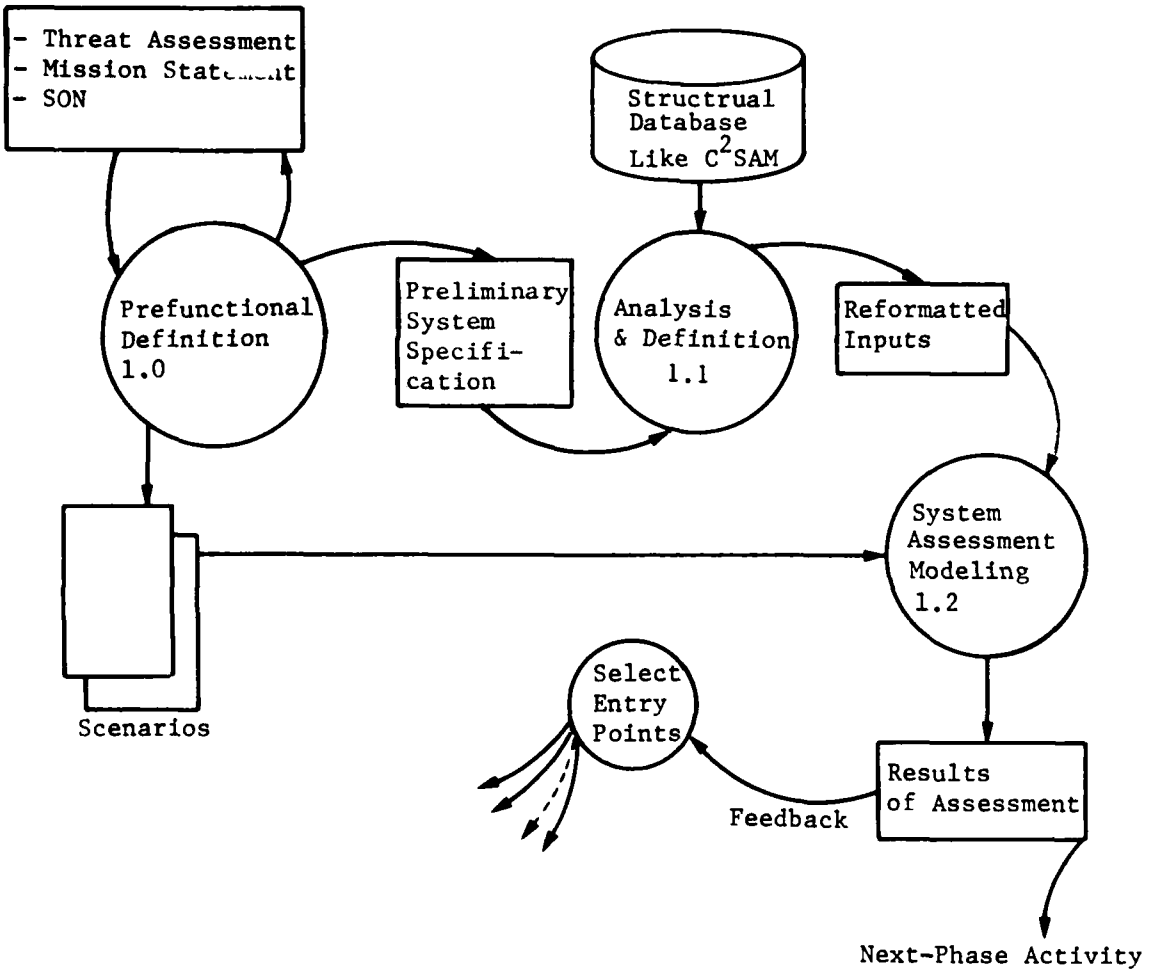


Figure 10.4-2 Express and Validate System Requirements

Phase 1: Express and Validate System Requirements

More specifically, the activities we are referring to in the Prefunctional Definition process include various types of high level operational analysis. Battle management, force allocation and assessment of threat are the kinds of activities involved. Experts in operations combine with researchers and consultants to do these analyses. The upper left-hand corner of Figure 10.4-2 is expanded in Figure 10.4-3 to show two activities taking place within Prefunctional Definition. Modeling and Scenario Development are the work of the operational experts. The translation activity takes their raw output to produce a Preliminary System Specification.

Figure 10.4-2 shows our highest level of division within the phase. Proceeding left to right, the Prefunctional Definition activity produces a preliminary system specification. This specification feeds the Analysis and Definition process, producing a more detailed and formatted specification. The formatted specifications are then used to model the system and assess it against scenarios.

Preliminary System Specification resulting from Prefunctional Definition drives the Analysis and Definition activity. Formatted specifications are a result. The Scenario Development by these same experts proceeds down the left side of the diagram. Detail in these scenarios increases with time. Arrows from the Scenarios back into the main diagram indicate that the Scenarios feed into all of the activities.

Scenarios provide a reference model for the state of the evolving system at any time. Figure 10.4-4 shows how Scenario development is a continuing process paralleling both the growth of detail in the actual system and the developing model of the system. As time goes on, all these increase in detail. Diagram 10.4-2 also shows a database such as the C2SAM as input to the functional analysis and definition activity. We are using the C2SAM here to represent a generalized functional model of C^3 systems. C2SAM itself contains both generic and specific information. Its data bases are validated for TAC, MAC, and USAFE (CENTCOM). The generic structure of C^2 systems has been examined and judged adequate for SAC and NORAD; therefore, the C2SAM paradigm can be extended to cover the necessary range of structures. Figure 10.4-3 expands the Analysis and Definition Activity to show the activities substructure and the categories of formatted specifications produced. These specifications are then reformatted to be input to the System Assessment modeling activity.

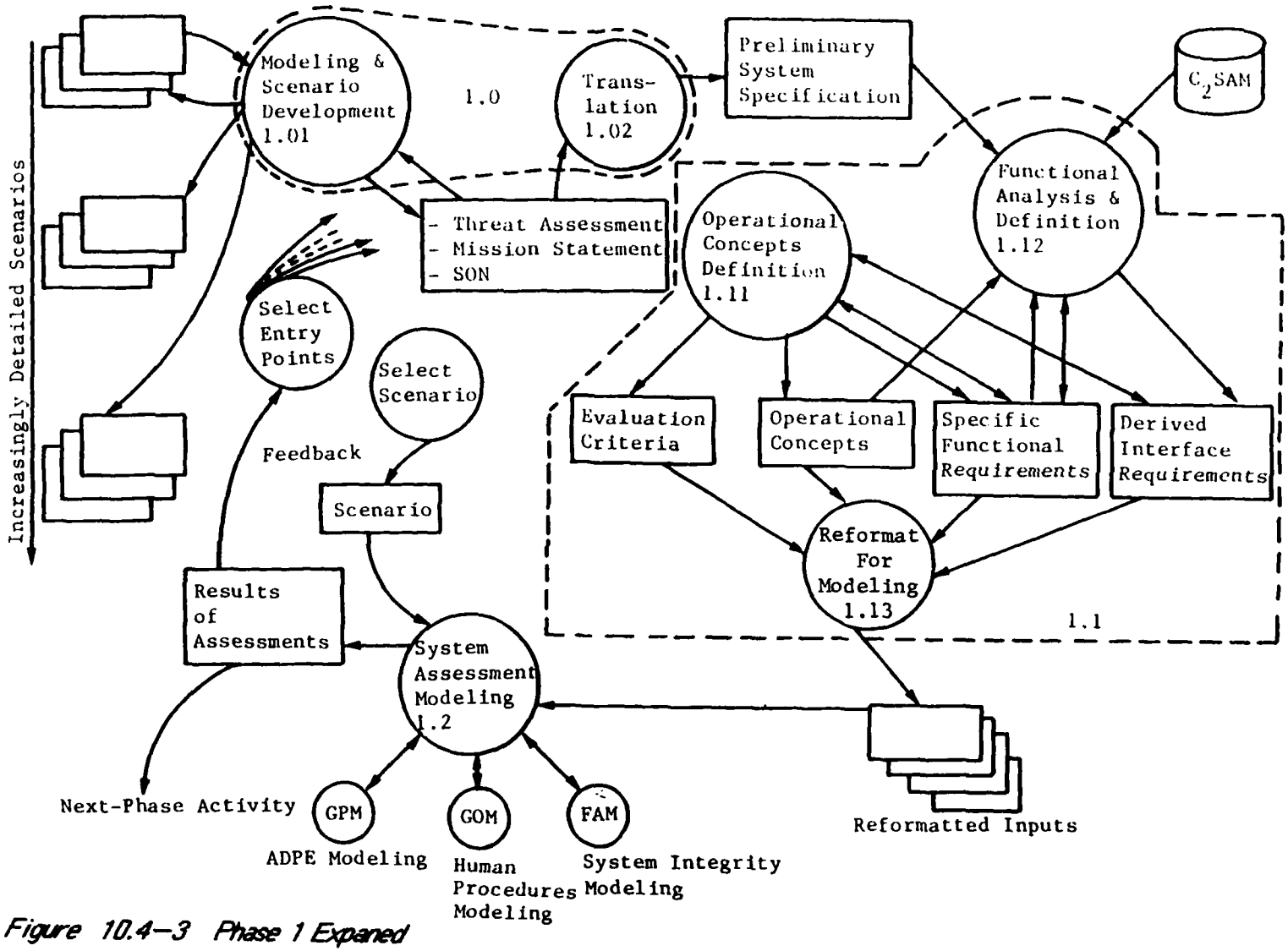


Figure 10.4-3 Phase 1 Expaned

Again, referring to Figure 10.4-3, it shows that System assessment modeling is achieved through the GPM, GOM, and FAM tools. The GPM provides ADPE modeling, the GOM allows modeling of human analyst procedures, and the FAM provides modeling of a system's logical structure.

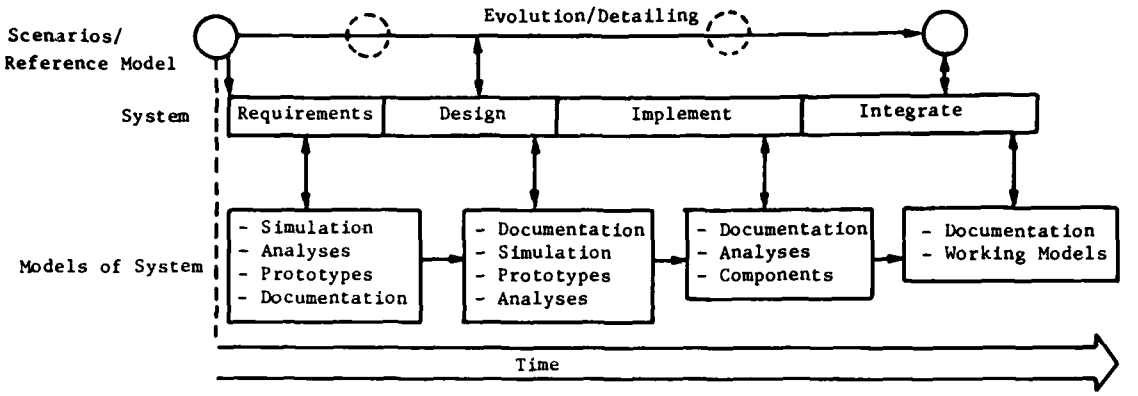


Figure 10.4-4
There are three continuing processes within a system development: coming-into-being of the product, development of paper and computer-based models, and evolution of the reference scenarios.

Figure 10.4-5 isolates the subactivities of functional analysis and definition. It shows the use of a generalized C^3 system functional model to arrive at specific functional requirements. The specific functional requirements are checked for consistency and completeness, and revisions are identified. Also, interface requirements are derived from the Specific Functional Requirements. The two products, Specific Functional Requirements and Derived Interface Requirements, are input to the Operational Concepts Definition. Operational Concepts Definition breaks down into the two activities depicted in Figure 10.4-6. First, operational concepts are developed from analysis. This set of concepts then yields a set of evaluation criteria for the system.

Phase 2: Express and Validate Software Requirements

Refer to Figure 10.4-7. Here we see the relationship of the project data bases to the methodology. They provide a repository into which the growing mass of detail in the system may be deposited. Their filling is done by the Software Requirements Analysis activity. The analysis takes preliminary software requirements and seeks to refine and clarify them. More detailed requirements are produced. The more refined requirements are modeled and the results of modeling used as input to further iteration of the analysis activity. Eventually, requirements are refined to the point that they may be prototyped. After prototypes are constructed and run, the results are fed into an activity which determines the best way to feed them back into the development process.

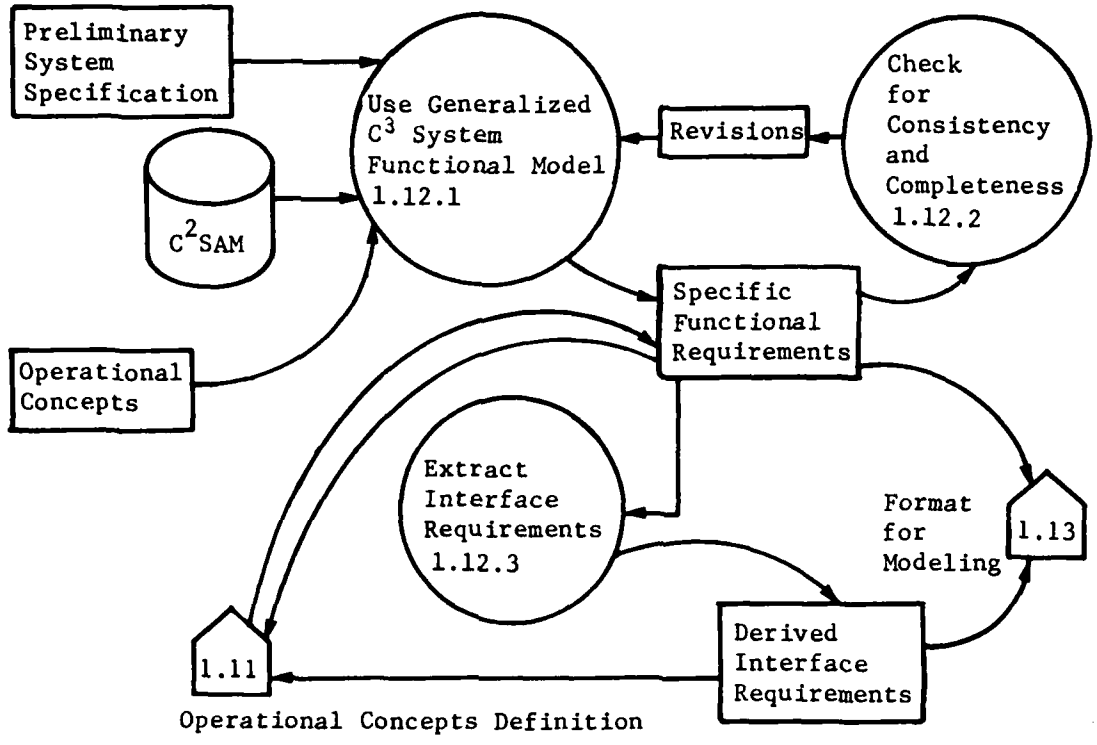


Figure 10.4-5 Functional Analysis and Definition

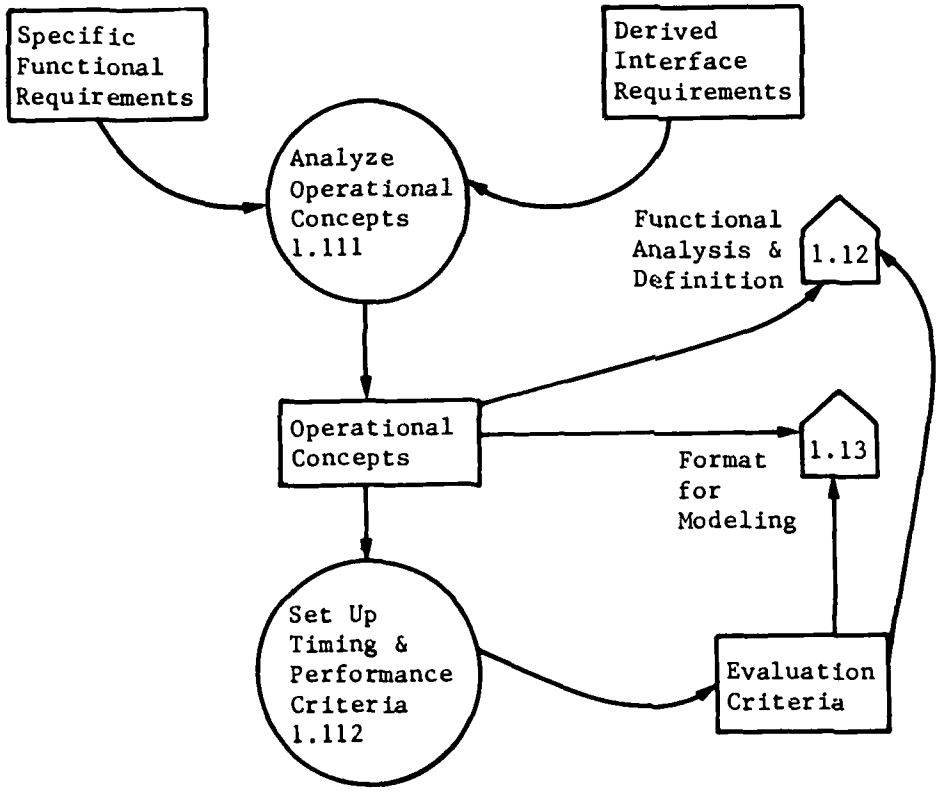


Figure 10.4-6 Operational Concepts Definition

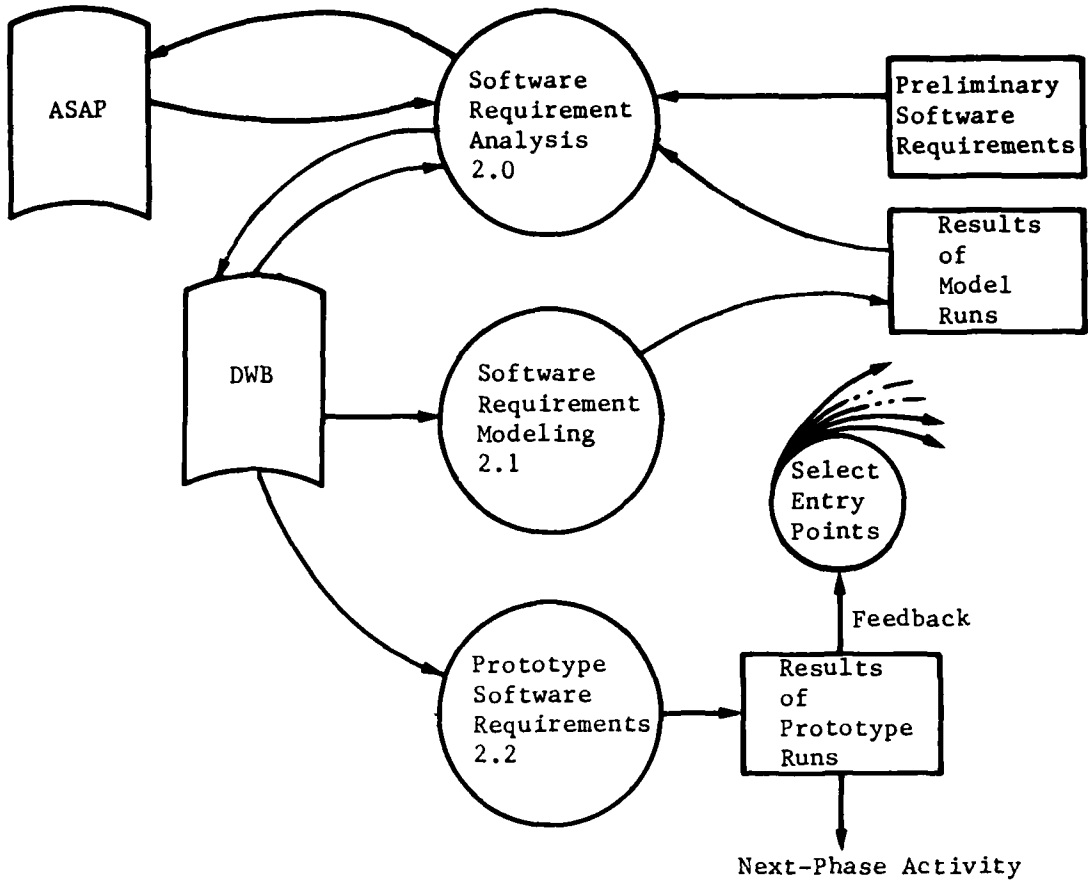


Figure 10.4-7 Express and Validate Software Requirements

Figure 10.4-8 expands on the Software requirements analysis activity. Preliminary Software Requirements initially feed the Site Dependent Analysis activity. Site dependent refers to methods of analysis peculiar to whatever organization is responsible for these activities. For instance, the method of tailored Yourdon analysis used by Denver Aerospace would be site dependent analysis. A DWB-derived data base is pictured as a storage place for the results of analysis. Two activities augment the site dependent analysis. One of these is Domain Dependent Analysis. Domain Dependent Analysis implies a collection of techniques developed specifically for the analysis of C^3I requirements, independent of site specific considerations. For instance, SREM and its supporting software specifically facilitate the isolation of statements of software requirements in the C^3I system developments. Again, we show a DWB defined data base as the repository of the results of such analysis. Also augmenting the analysis process is the structural analysis activity. We represent this type of analysis by the use of the ASAP tool. ASAP, a Denver Aerospace tool, supports a structured analysis technique. The technique consists of expressing the requirements in input-process-output terms and verifying that the logical flow is consistent and complete. The

requirements are considered consistent if subrequirements trace to higher level requirements in terms of specific inputs and outputs. Completeness implies that all inputs and outputs are defined, each required process has some input, and all output has an origin. The results of this analysis feed back into the analysis activity. The end results of all types of analysis are formatted requirements stored in a DWB-derived data base.

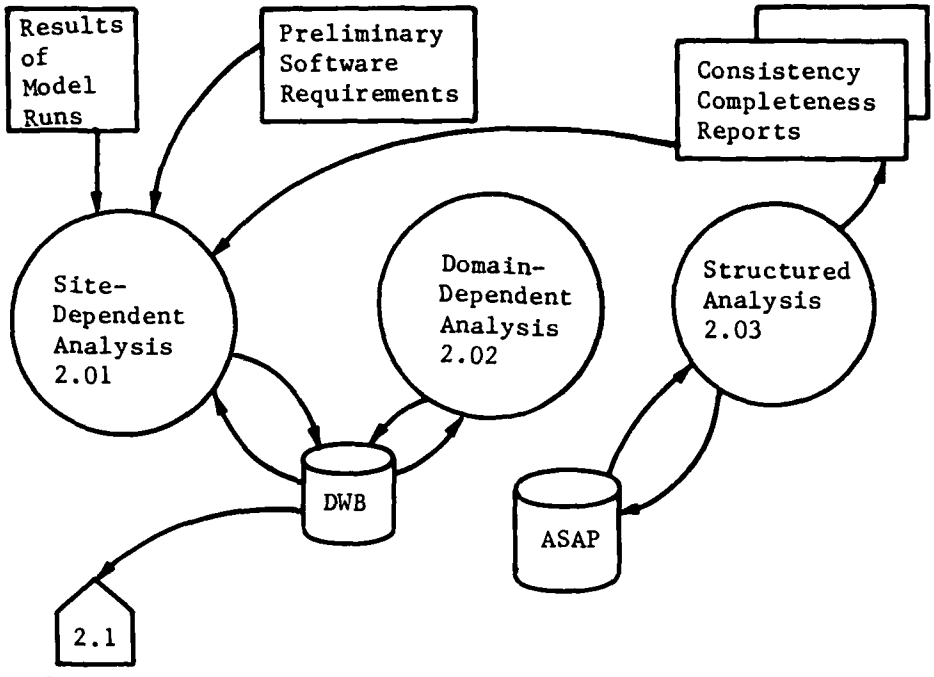


Figure 10.4-8 Software Requirement Analysis

Refer now to Figure 10.4-9. The next activity, Software Requirement Modeling, shows that the requirements are obtained from the data base. Technicians translate the requirements into descriptions suitable for such models as FAM, GOM and GPM. A test scenario is selected and the models are run. The software requirements analysis activity then considers the results.

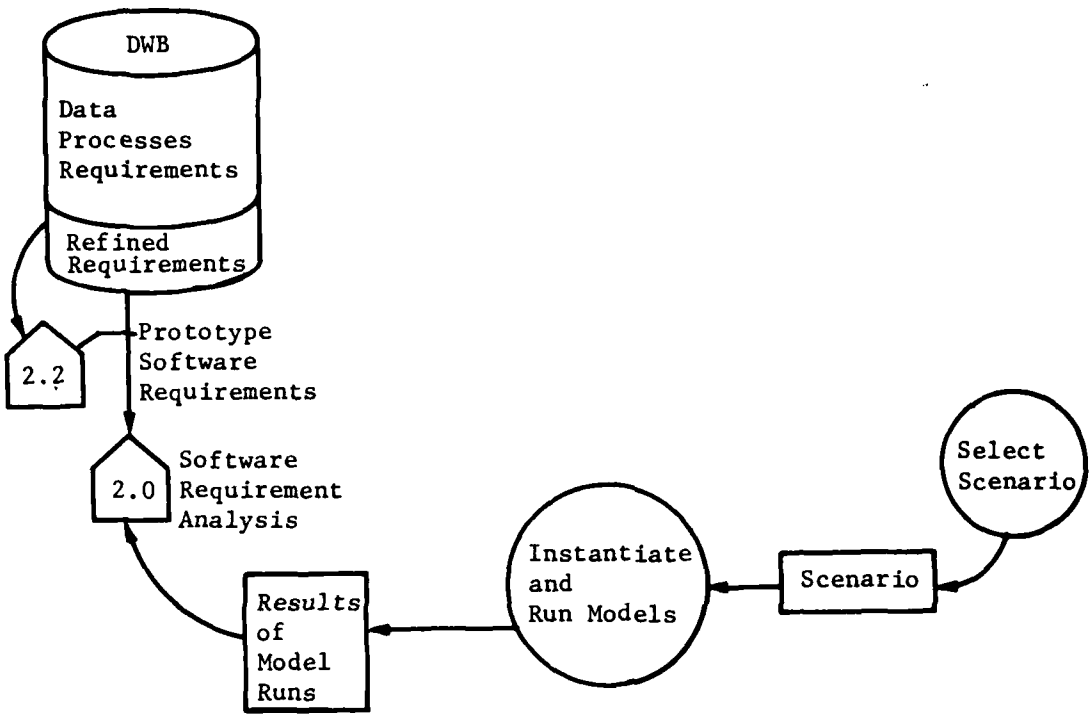


Figure 10.4-9 Software Requirement Modeling

The analysis and modeling activities accumulate a set of refined requirements in the DWB. When sufficient refined requirements have accumulated and the situation warrants, prototyping of the requirements begin.

Figure 10.4-10 expands the Prototype Software Requirement Activity. As the diagram shows, prototyping activities draw on the set of refined requirements stored in DWB. These activities utilize currently used methods. These methods consist of prototyping environments which require derivations of initial requirements. In this case, the initial requirements are the refined requirements stored in DWB. From these, subrequirements necessary to instantiate prototypes must be derived. Two branches emerge from the subrequirement derivation. One branch leads to the implementation of C^3I MMI prototypes. The other branch implements unspecified types of functional and performance prototypes.

At the end of the high and low risk paths, the prototypes are run using our appropriate scenario. Results of the runs are fed into a feedback

selection activity to determine the best point or points in the process to send results to.

Phase 3; Software Design

We have not focused a great deal on this final phase. To begin with, much effort has already addressed this level of activity. Methodology exists and is well known for getting from well-defined and detailed requirements to software designs and then to executable code. We can achieve the highest payoffs by focusing on producing good inputs for this phase. By good inputs we mean well-defined, validated requirements.

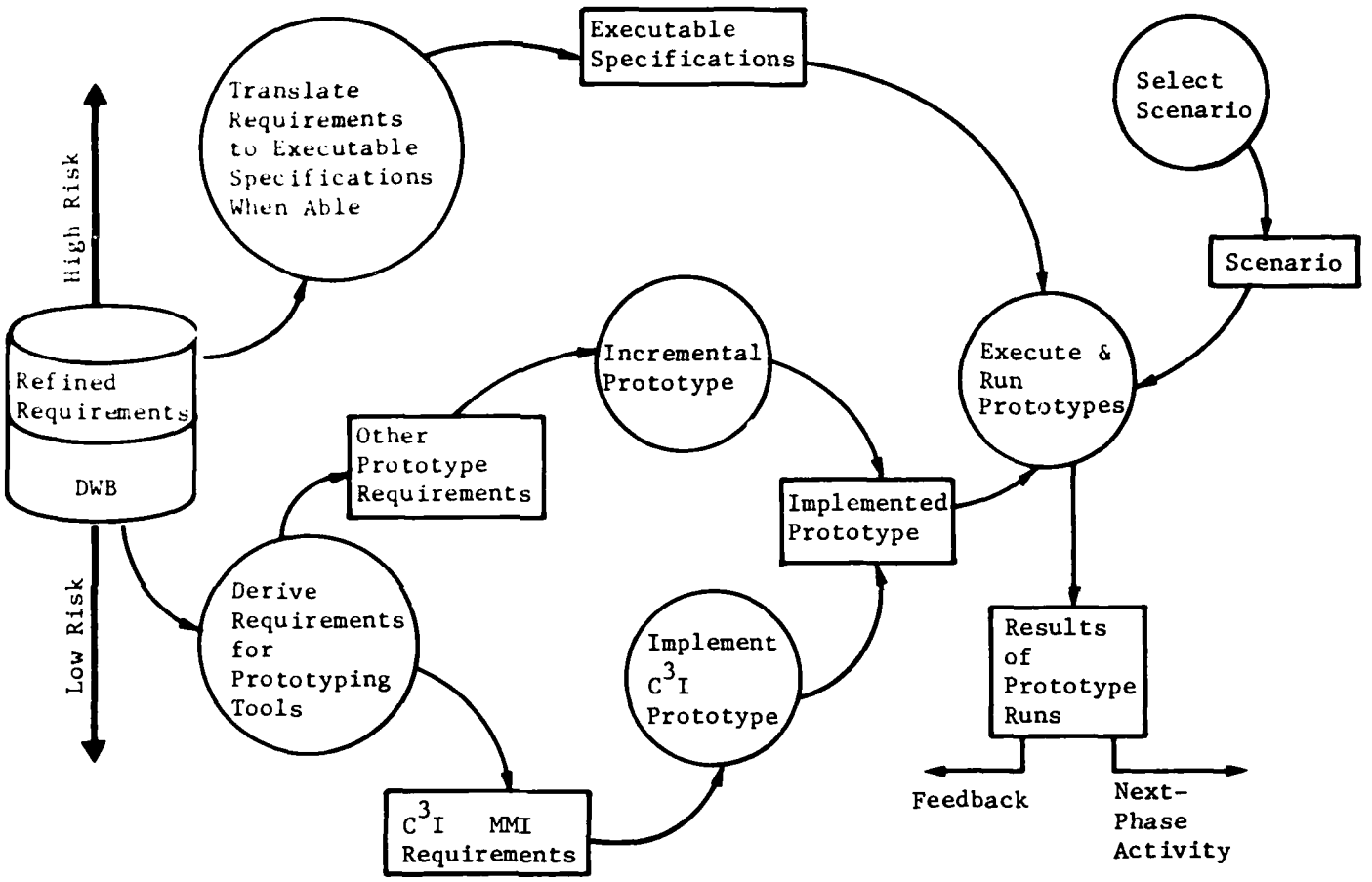


Figure 10.4-10 Prototype Software Requirements

Figure 10.4-11 details Phase 3. A DWB derived database again serves as the main repository of specifications. Here we have partitioned the database into several layers. The top layer holds validated software requirements. The first activity expands the required process description in text producing the next lower level of specification. Then the expanded descriptions are translated into some form of PDL (Program Design Language). Finally, the PDL descriptions are translated to executable form.

The executable forms are prototypes. Selected scenarios drive the prototypes, producing results which are evaluated against high level design criteria accumulated throughout the development cycle. Analysis of the results reveals the state of the system in relation to the established criteria. A selection activity decides where in the cycle to direct the evaluations.

10.5 Scenario Library

In terms of the C^3I Rapid Prototype Investigation's own methodology, it is at a point which allows filling in enough details of a system prototype to make some 'runs' and validate some system requirements. Scenarios must be developed to drive the prototype. To do so, we can draw on the extensive record of C^3I system development and the knowledge of operationally experienced people. Prime candidates for these scenarios were identified early on in the project and remain viable, the OASIS and SPADOC efforts in particular. There is detailed history of these efforts, encompassing all levels of development to some extent. Formal documentation, developed software, and participants in the efforts all contribute to the legacy and can be used to derive scenarios.

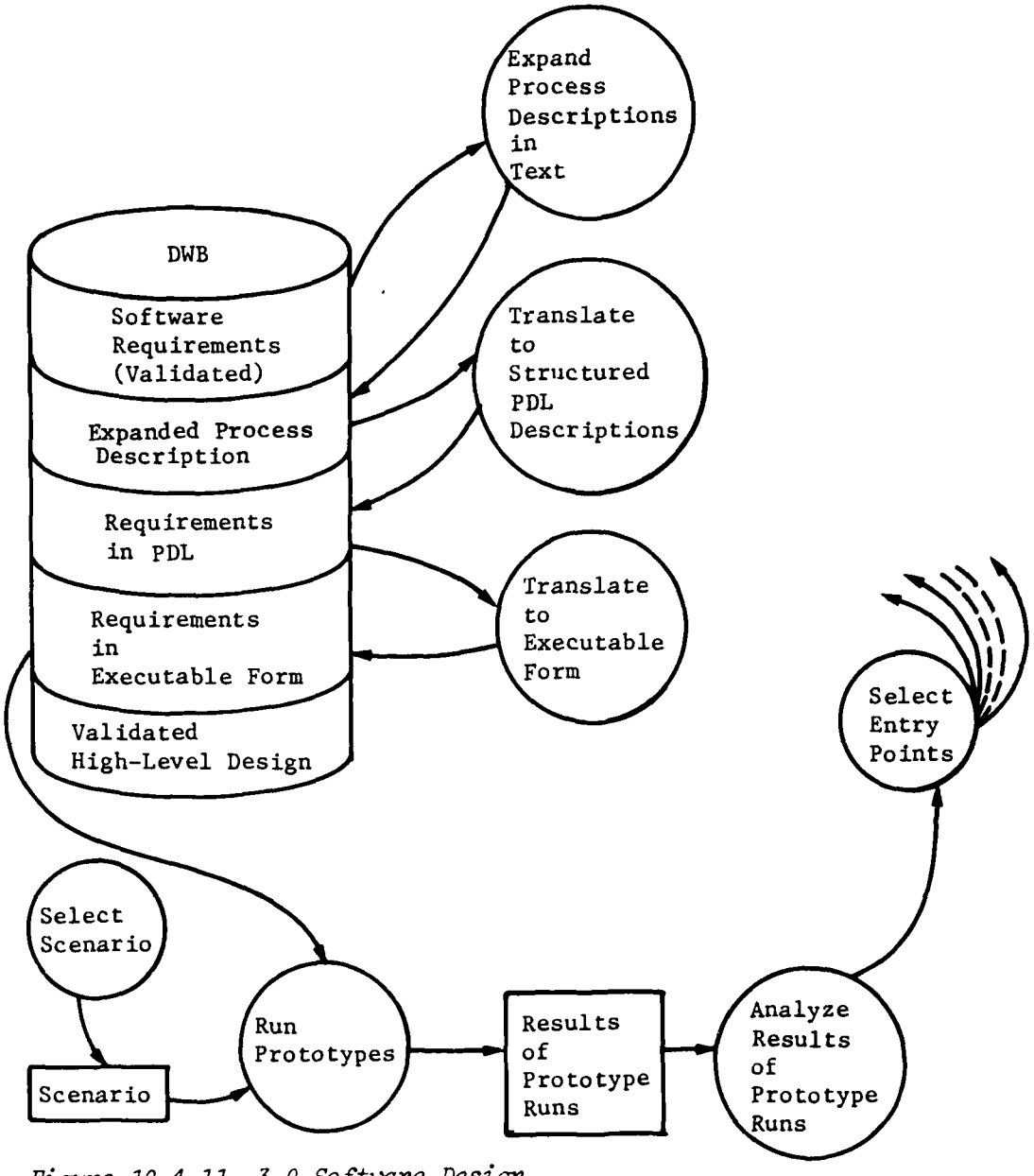


Figure 10.4-11 3.0 Software Design

11.0 PROTOTYPE ENVIRONMENT DESCRIPTION

The Rapid Prototyping System (RPS) will be a tool to prototype key functions of Air Force C^3I systems. The RPS development will be primarily of software. Figure 11-1 shows the major CPC components and their interconnection. The RPS consists of 6 CPCs: four modeling CPCs and two tools/support CPCs. The user will access the system through the master control CPC and, generally, next work with the interface modeling CPC. The master tools CPC allows coordination of the other modeling CPCs and their packages. At any point, the user may direct text or data to the support tools CPC to support documentation. The RPS will provide means to characterize the user-system interface aspects of a C^3I system, its data base content and conceptual design, its structure at varying levels of detail, and its expected performance. The RPS will support documentation of the results as well as storage and retrieval of the prototypes themselves. At some future time, links to other systems and models may be required.

Figure 11-1 shows an equipment environment of the RPS. It consists of a Digital Equipment VAX 11/780 computer with two disk drives, line printer, system terminal, at least 2 MBYTES main memory, tape drive and at least two VT100 terminals. There will also be a workstation network interfaced to the VAX 11/780 unibus through a gateway processor (Apollo DSP 80 or functional equivalent). There will be a machine to process Prolog which has at least 2 MBYTES main memory and approximately 1 GBYTES of peripheral storage. System engineering terminals on the order of IBM PC or Apple MacIntosh's will furnish database query response and text editing capabilities. There should be included color workstations such as the Apollo DN 550. This workstation should have at least 2 MBYTES main memory and several MBYTES of display memory. Each monochrome workstation (DN 320) will have its own Winchester storage of at least seventy magabytes. The DSP 80 will be connected to a dual drive removable cartridge Winchester system of at least 10 MBYTES per cartridge capacity. There will be a screen hardcopy device interfaced to the Domain network. The VAX 11/780 will be interfaced to the RADC fiber optics network.

The VAX 11/780 shall host the VMS operating system, FORTRAN, Prolog, PASCAL and C compilers. The Apollo DN 320S should be provided with the Aegis operating system, FORTRAN, Prolog, PASCAL, and C compilers. The Distributed Software Engineering Environment (DSEE) and Domain Distributed Data Management package (D^3M) should be provided for the Apollo computers. Further details of the software functions may be found in the RPS Functional, System/Subsystem, and Program Specifications.

Rapid Prototyping System Hardware

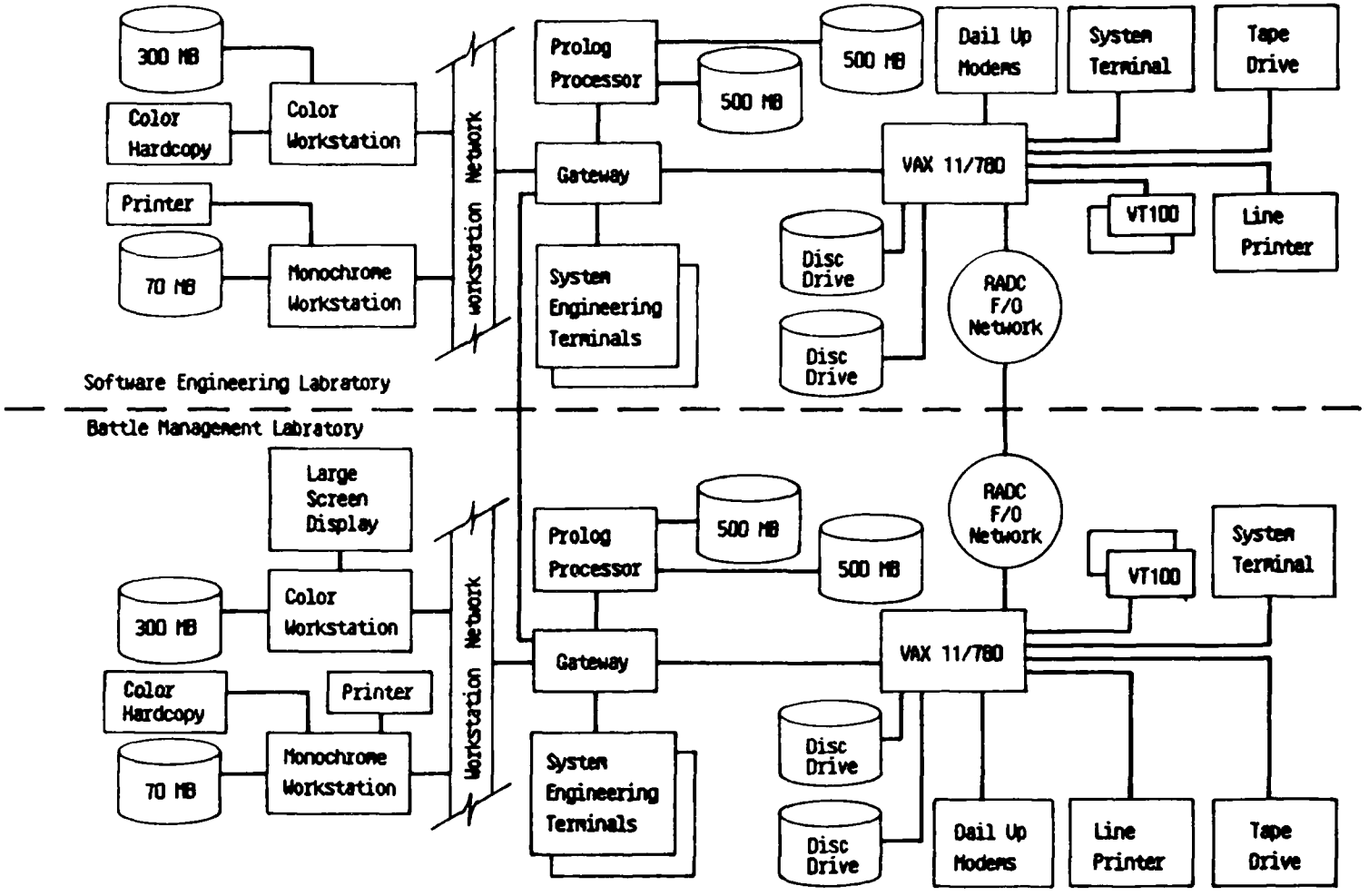


Figure 11-1

The RPS Software Components

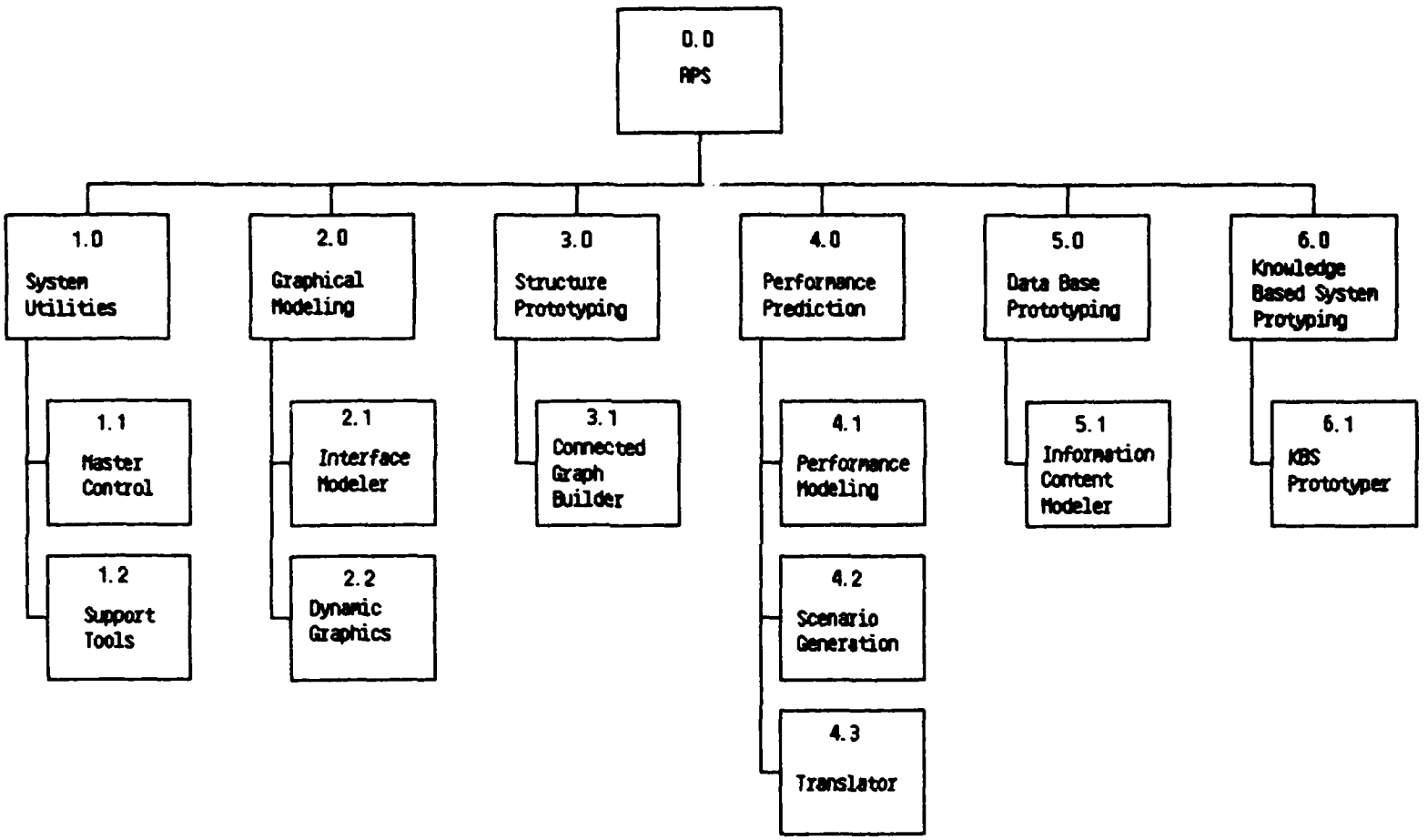


Figure 11-2

12.0 DEMONSTRATION SCENARIO

12.1 Introduction

We held three demonstrations at Martin Marietta Denver Aerospace to examine the suitability of our prototyping tools. The key demonstration was focused on a C^3I center that emphasized displays and databases. The Ground Attack Control Center (GACC) is a concept for an improved element of the Tactical Air Control Center (TACC). It was our intention to pick a demonstration problem representative of those requiring rapid prototyping. We chose the GACC because it is currently under definition, it will be a system capable of development within the next five years, and it includes elements such as displays and databases.

The intention was to present an element of a GACC as an example of prototyping--not to define or solve a GACC problem. The exact scenario we used was incidental. We assigned a prototypical mission user the job of creating the GACC prototype. He had virtually no software experience. He developed a scenario and used our prototype tools to construct a demonstration. This activity was supposed to validate the ease of using rapid prototyping tools as well as validating different types of prototyping. We used interface prototyping and structure prototyping to create the TACC demonstration.

12.2 Demonstration Results

The GACC demonstration proved the utility of our prototype rapid prototyping tool and approach. Key to any software system designed to support human cognitive processing is the user interface. User satisfaction is vested in this interface's functionality. We had hypothesized that maximal reconfigurability of the USI would be useful. Major steps were taken toward that goal. The GACC prototype USI was assembled quickly (two-three week period) and was easily reconfigured to allow a one day turnaround of changes. It should be emphasized that this was accomplished with a partial and incomplete interface modeling tool. Definition, placement, and selectivity of pop-up menus and icons took place. The level of detail in the definition of the GACC USI was sufficient to have provided valuable support at a system requirements review, operational concept review, or preliminary design review. Issues such as concerned the type of data needed to be included in the database for a GACC were exposed. Operational procedures enabling the prototype analyst to perform the experiment mission were apparent. The prototype analyst with no programming experience was able to build the GACC USI himself with minimal aid from technical specialists. This aid was of an advisory nature. The USI states could be captured in hardcopy and would have been available for inclusion in specifications if they were to be built.

This demonstration provided conclusive evidence of the utility of a rapid prototyping system based on modeling to support a key and crucial aspect of C^3I system development.

APPENDIX

```
MODULE EXPR VAL SYSTEM REQS IS
      ATTRIBUTES          COMPUTERIZED, MANUAL:

      INPUT DATA          C2SAMDB, MISSION STATEMENT, THREAT ASSESSMENT,
                          A-SPEC,
                          SON:

      OUTPUT DATA         SPECIFIC FUNCTIONAL REQUIREMENTS,
                          DERIVED PERFORMANCE REQUIREMENTS,
                          PRELIMINARY SYSTEM SPECIFICATION,
                          DERIVED INTERFACE REQUIREMENTS,
                          OPS CONCEPTS,
                          REFORMATTED INPUTS,
                          SCENARIOS,
                          SYSTEM ASSESSMENT:

      INITIALIZES         REFORMATTED INPUTS,
                          SCENARIOS,
                          SYSTEM ASSESSMENT:

      MODIFIES            OPS CONCEPTS,
                          SPECIFIC FUNCTIONAL REQUIREMENTS,
                          DERIVED PERFORMANCE REQUIREMENTS,
                          REFORMATTED INPUTS:

      ACTIVATES           EXPR VAL SOFTWARE REQS:

      CONSISTS OF         MODELING SCENARIO DEVELOPMENT,
                          PRELIM SYS SPECIFICATION
                          DERIVE FUNCTIONAL REQS,
                          CHECK CONSISTENT COMPLETE,
                          OPS CONCEPT ANALYSIS,
                          EXTRACT INTERFACE REQS,
                          BUILD EVALUATION CRITERIA,
                          FORMAT FOR MODELING,
                          SYSTEM ASSESSMENT MODELING,
                          HUMAN PROCEDURES MODELLING,
                          SYSTEM INTEGRITY MODELING:

      TITLE               EXPRESS AND VALIDATE SYSTEM REQUIREMENTS:

      DESCRIPTION         THIS IS THE FIRST PHASE OF DEVELOPMENT IN THE
                          RAPID PROTOTYPING CYCLE. IT BEGINS THE PROCESS OF
                          ITERATING OVER ALTERNATIVE PROTOTYPES OF THE
                          SYSTEM BY DERIVING MODELLABLE SPECIFICATIONS FROM
                          THE INITIAL SYSTEM DOCUMENTATION, SON, MISSION
                          STATEMENT, ETC., TRANSLATING THE MODELLABLE
                          SPECIFICATIONS INTO A PROTOTYPE, EXERCISING THE
                          PROTOTYPE AND ITERATING OVER PREVIOUS STEPS AS
                          NECESSARY AND AS INDICATED BY RESULTS OF PROTOTYPE
                          RUNS.

                          OUTPUTS FROM THIS PHASE ARE FED BACK INTO VARIOUS
                          COMPONENTS OF THE PHASE AND INTO THE MORE DETALED
                          PHASE 'EXPRESS AND VALIDATE SOFT ARE
                          REQUIREMENTS'.;
```

SPECIFICATION

INPUT :

2SAMUB, MISSION STATEMENT, THREAT ASSESSMENT,
A SPEC,
SON

REPEAT

DO MODELLING SCENARIO DEVELOPMENT
USING :
MISSION STATEMENT, THREAT ASSESSMENT AND
STATEMENT OF NEED
PRODUCING :
SCENARIOS

DO PRELIM SYSTEM SPECIFICATION
USING:
MISSION STATEMENT, THREAT ASSESSMENT AND
STATEMENT OF NEED
PRODUCING :
PRELIMINARY SYSTEM SPECIFICATION

DO IN PARALLEL
* These parallel processes are to proceed *
* as the necessary inputs become available *

DERIVE FUNCTIONAL REQS
USING:
PRELIMINARY SYSTEM SPECIFICATION,
C2SAMDH,
CONSISTENCY AND COMPLETENESS CHECKS AND
OPS CONCEPTS
PRODUCING:
SPECIFIC FUNCTIONAL REQUIREMENTS

AND

OPS CONCEPT ANALYSIS
USING:
SPECIFIC FUNCTIONAL REQUIREMENTS AND
DERIVED INTERFACE REQUIREMENTS
PRODUCING:
OPS CONCEPTS

AND

CHECK CONSISTENT COMPLETE
USING:
SPECIFIC FUNCTIONAL REQUIREMENTS
PRODUCING:
CONSISTENCY AND COMPLETENESS CHECKS

```
AND

  EXTRACT INTERFACE REQS
    USING:
      SPECIFIC FUNCTIONAL REQUIREMENTS
    PRODUCING:
      DERIVED INTERFACE REQUIREMENTS

AND

  BUILD EVALUATION CRITERIA
    USING :
      OPS CONCEPTS
    PRODUCING:
      PERFORMANCE AND TIMING EVALUATION
      CRITERIA

AND

  FORMAT FOR MODELLING
    USING :
      DERIVED INTERFACE REQUIREMENTS AND OPS
      CONCEPTS
    PRODUCING :
      REFORMATTED INPUTS

AND

  SYSTEM ASSESSMENT
    USING :
      PREFORMATTED INPUTS,
      PERFORMANCE AND TIMING EVALUATION
      CRITERIA,
      AND MODEL RESULTS
    PRODUCING :
      FEEDBACK

AND

  ADPE MODELLING
    USING:
      REFORMATTED INPUTS,
      PERFORMANCE AND TIMING EVALUATION
      CRITERIA
      AND MODEL RESULTS
    PRODUCING :
      MODEL RESULTS

AND

  HUMAN PROCEDURES MODELLING
    USING :
      REFORMATTED INPUTS,
      PERFORMANCE AND TIMING EVALUATION
      CRITERIA,
```

```
                    AND MODEL RESULTS
                  PRODUCING :
                    MODEL RESULTS

              AND

                SYSTEM INTEGRITY MODELLING
                  USING:
                    REFORMATTED INPUTS,
                    PERFORMANCE AND TIMING EVALUATION
                    CRITERIA,
                    AND MODEL RESULTS
                  PRODUCING:
                    MODEL RESULTS

            END IN PARALLEL
            UNTIL CRITERIA SATISFIED

            OUTPUT :
                SPECIFIC FUNCTIONAL REQUIREMENTS,
                DERIVED PERFORMANCE REQUIREMENTS,
                PRELIMINARY SYSTEM SPECIFICATION,
                DERIVED INTERFACE REQUIREMENTS,
                OPS CONCEPTS,
                REFORMATTED INPUTS,
                SCENARIOS,
                SYSTEM ASSESSMENT


END EXPR VAL SYSTEM REQS;

MODULE EXPR VAL SOFTWARE RENS IS

      ATTRIBUTES        CONPUTERIZED, MANUAL;

      INPUT DATA        PRELIMINARY SOFTWARE REQS;

      OUTPUT DATA       ASAP DATABASE, DWB DATABASE,
                        CONSISTENCY COMPLETENESS CHECKS
                        TRANSLATED RESULTS AND NEW REQUIREMENTS;

      INITIALIZES       ASAP DATABASE, DWB DATABASE,
                        CONSISTENCY COMPLETENESS CHECKS,
                        TRANSLATED RESULTS AND NEW REQUIREMENTS;

      MODIFIES          ASAP DATABASE, DWB DATABASE,
                        CONSISTENCY COMPLETENESS CHECKS,
                        TRANSLATED RESULTS AND NEW REQUIREMENTS;

      ACTIVATES         SOFTWARE DESIGN;
```

```
CONSISTS OF       SITE DEPENDENT ANALYSIS,
                  DOMAIN DEPENDENT ANALYSIS,
                  BUILD DATABASES,
                  CHECK CONSISTENT COMPLETE,
                  ASSESS SW REQS,
                  GENERATE NEW REQS,
                  PROTO NO NEW REQS,
                  GEN MMI PROTOS,
                  GEN OTHER PROTOS,
                  EXECUTE ASSESS PROTOS,
                  SELECT SCENARIOS;

TITLE             EXPRESS AND VALIDATA SOFTWARE REQUIREMENTS;

DESCRIPTION       THIS PHASE OF THE RAPID PROTOTYPING PROCESS BEGINS
                  AFTER THE INITIAL PHASE 'EXPRESS AND VALIDATE
                  SYSTEM REQUIREMENTS' HAS PRODUCED TENTATIVE
                  SOFTWARE ALLOCATIONS IN THE FORM OF PRELIMINARY
                  SOFTWARE REQUIREMENTS.  IN THIS PHASE DETAILED
                  SOFTWARE MODELLING IS PERFORMED AND THE RESULTS
                  USED TO VALIDATE AND DIRECT THE SOFTWARE DESIGN
                  PROCESS.

SPECIFICATION     INPUT :
                    PRELIMINARY SOFTWARE REQUIREMENTS

                  DO IN PARALLEL

                    *     These parallel processes are to be *
                    *   initiated as necessary inputs become *
                    *   available and schedule permits.

                      SITE DEPENDENT ANALYSIS
                        USING:
                          PROTOTYPE RESULTS,
                          PRELIMINARY SOFTWARE REQUIREMENTS AND
                          CONSISTENCY COMPLETENESS CHECKS
                        PRODUCING :
                          SITE ANALYSIS

                      AND

                        DOMAIN DEPENDENT ANALYSIS
                          USING :
                            SITE ANALYSIS AND
                            SOFTWARE REQ ASSESSMENT
                          PRODUCING :
                            DATA BASE INPUTS

                      AND

                        BUILD DATABASES
                          USING :
                            DATABASE INPUTS AND
```

```
      PROTOTYPE ASSESSMENT
    PRODUCING :
      ASAP DATABASE AND
      DWB DATABASE

AND

  ASSESS SW REQS
    USING :
      ASAP DATABASE,
      SWB DATABASE AND
      MODEL RESULTS
    PRODUCING :
      MODEL CONFIGURATIONS AND
      SOFTWARE REQ ASSESSMENT

AND

  RUN MODELS
    USING :
      MODEL CONFIGURATIONS
    PRODUCING :
      MODEL RESULTS

AND

  PROTO NO NEW REQS
    USING :
      ASAP DATABASE,
      DWB DATABASE
    PRODUCING
      TRANSLATED RESULTS

AND

  GENERATE REQS
    USING :
      ASAP DATABASE AND
      DWB DATABASE
    PRODUCING:
      NEW REQUIREMENTS AND
      TRANSLATED RESULTS

AND

  GEN MMI PROTOS
    USING :
      NEW REQUIREMENTS
    PRODUCING :
      MMI PROTOTYPE

AND
```

```
                    GEN OTHER PROTOS
                      USING :
                        TRANSLATED RESULTS
                      PRODUCING :
                        FUNCTIONAL PROTOTYPE AND
                        PERFORMANCE PROTOTYPE

                  AND

                    EXECUTE ASSESS PROTOS
                      USING :
                        SCENARIO SELECTION,
                        MMI PROTOTYPE,
                        FUNCTIONAL PROTOTYPE AND
                        PERFORMANCE PROTOTYPE
                      PRODUCING :
                        PROTOTYPE ASSESSEMENT

                  AND

                    SELECT SCENARIO
                      USING :
                        SCENARIOS
                      PRODUCING :
                        SCENARIO SELECTION

              END IN PARALLEL

              OUTPUT :
                  ASAP DATABASE, DWB DATABASE,
                  CONSISTENCY COMPLETENESS CHECKS,
                  TRANSLATED RESULTS AND NEW REQUIREMENTS:

END EXPR VAL SOFTWARE REQS

MODEL SOFTWARE DESIGN IS

      ATTRIBUTES        COMPUTERIZES, MANUAL;

      INPUT DATA        DWB DATABASE;

      INITIALIZES       PROCESS DESCRIPTED REQS,
                        PDL REQS,
                        EXECUTABLE REQS,
                        VALIDATED HIGH LEVEL DESIGN;

      MODIFIES          PROCESS DESCRIPTED REQS,
                        PDL REQS,
                        EXECUTABLE REQS,
                        VALIDATED HIGH LEVEL DESIGN:

      DESCRIPTION       THIS PHASE BEGINS WITH VALIDATED SOFTWARE
                        REQUIREMENTS, TRANSLATES THOSE REQUIREMENTS INTO
                        ANEXECUTABLE FORM (MODEL CONFIGURATION), EXECUTES
                        PROTOTYPES AND ANALYZES THE RESULTS.:
```

```
SPECIFICATION:          INPUT:
                          DWB DATABASE

                        REPEAT

                          DO EXPAND PROCESS DESCRIPTORS
                              USING :
                                VALIDATED SW REQS
                              PRODUCING :
                                PROCESS DESCRIPTED REQUIREMENTS

                          DO TRANS TO PDL
                              USING :
                                PROCESS DESCRIPTED REQUIREMENTS
                              PRODUCING :
                                PDL REQUIREMENTS

                          DO TRANS TO EXECUTABLE
                              USING :
                                PDL REQUIREMENTS
                              PRODUCING:
                                EXECUTABLE REQUIREMENTS

                          DO SELECT SCENARIO
                              USING :
                                SCENARIOS
                              PRODUCING :
                                SCENARIO SELECTION

                          DO RUN PROTOTYPES
                              USING :
                                SCENARIO SELECTION AND
                                EXECUTABLE REQUIREMENTS
                              PRODUCING :
                                PROTOTYPE RESULTS

                          DO ANALYZE RESULTS
                              USING :
                                PROTOTYPE RESULTS
                              PRODUCING :
                                RESULTS ANALYSIS

                        UNTIL CRITERIA MET;

END SOFTWARE DESIGN:
```

REFERENCES

The references are in alphabetical order by title, and the numbers which occur are the C^3I Rapid Prototype Investigation library index numbers.

23. A3 - Affordable, Acquisition, Approach (attach C); Ltg. Stewart

10. AFCEA - C2 System Acquisition Study - Final Briefing 16 July 1976

11. AFCEA - C2 System Acquisition Study - Final Report 01 Sept. 1982

9. AFR 57-4 Operational Requirements Modification Program Approval 15 Dec. 1977

8. AFR 80-14 Research and Development - Test and Evaluation 19 July 1976

31. AFR 300-2 Data Automation, Managing the USAF Automated Data Processing Program 24 April 1980

30. AFR 800-2 Acquisition Management, Program Management; Department of the Air Force 13 Aug. 1982

7. AFR 800-14
Vol. I - Management of Computer Resources in Systems
Vol. II - Acquisition and Support Procedures for Computer Resources in Systems 12 Sept. 1975

160. "Ace: A System Which Analyses Complex Explanations"; International Journal of Man-Machine Studies, Jan. 1979, Volume 11 #1, pg 125

216. "Ada Software Development Tools Up"; Electronics, May 1983, pg 157

248. "An Adaptable Software Environment to Support Methodologies"; IEEE Softtair Proceedings, July 25-28, 1983, pg 363

212. "Advanced Parallel Architectures Get Attention as Way to Faster Computing"; Electronics, June 1983, pg 105

315. "Another Program for Drawing Diagram"; Software-Practice and Experience, May 1982, Volume 12, Issue #5, pg 397

285. APL as a Software Design Specification Language; The Computer Journal, August 1980, Volume 23 #3, pg 230

116. "Application Methodology and Discussions" (Research Directions in Software Technology) 1980; R.O. Duda, R. Schank, M. Hammer

112. "Applications of Mathematical System Theory to System Design, Modeling and Simulation"; A. Wayne Wymore, Ph.D. (1981 Winter Simulation Conference Proceedings)

66. "The Art of Natural Graphic Man-Machine Conversation"; James D. Foley April 1974

109. "Artificial Intelligence--Applied to C^3I"; David A. Brown and Harvey S. Goodman

75. "Automated Documentation System User's Manual" H. Sayani, B. Kahn, and M. Zenn ISDOS Project July 1975

100. "Axioms for User-Defined Operators"; I.C. Pyle July 1979

36. Bill Batz Martin Marietta Aerospace Memo 9 May 1983

33. "Breaking The Systems Development Bottleneck"; Lee L. Gremillion and Philip Pyburn March/April 1983

345. "Building Control Structures in the Smalltalk-80 Systems"; L. Peter Deutsch, Smalltalk-80 System--Byte Magazine August 1981

342. "Building Data Structures in the Smalltalk-80 Systems"; James C. Althoff Jr., Smalltalk-80 System--Byte Magazine August 1981

21. C2 Software Acquisition and Development Working Group (Final Report); Chairman, Mr. Victor E. Jones July 1980

20. C2 Software Development and Acquisition Study - Status Report; Harry Kottcamp (W/28)

361. C3CM Structural Design--Appendix A: Battle Management Processor Software Requirements; Appendix B: Par System Development Methodology, Dec. 1982, (Technical Proposal)

19. C^3I Lecture Series (Command, Control, Communications and Intelligence) (Mire) March 1983

18. "C^3I Recommendations"; Alan J. Roberts (Mitre) July 1982

16. C^3I Systems Research and Evaluation Library - Vol. I (Mitre) Nov. 1982

17. C^3I Systems Research and Evaluation Library - Vol. II (Mitre) Nov. 1982

166. "A Case Study in Rapid Prototyping"; Software-Practice and Experience, 1980, Volume 10, pg 1037

215. "Case Study: SLQ-32 Design-to-Price EW Software Poses Expensive Challenge"; Defense Electronics, January 1983, pg 84.

99. "A Centralized Design Support Center"; Bruce Duncan March 1979

226. "The Challenge of Software Engineering Project Management"; Computer, August 1980, Volume 13, pg 51

193. "Challenges in Software Development"; Computer, March 1983, pg 60

12. Comments Arising From Lt. Col. Herndon; Harry Kottcamp

351. "Command Centers Have a Whole New Look (Command and Control)"; Military Electronics, April 1983, pg 22

81. "The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems"; Thomas P. Moran March 1981

199. "Communications Sequential Processes"; Communications of the ACM, Jan. 1983, Volume 25 #1, pg 100

133. "Compilation of Nonprocedural Specification Into Computer Programs"; IEEE Transactions on Software Engineering, May 1983, Volume SE-9 #3, pg 267

71. "Computer-Aided Software Development"; Daniel Teichroew, Hershey III, and Yamamoto May 1977

186. "Computer-Aided Production of Language Implementation Systems"; Software-Practice and Experience, Sept. 1982, Volume 12, Issue #9, pg 785

227. "Computer Aided Programming (Part I)"; IEEE Softfair Proceedings, July 25-28, 1983, pg 9

259. "Computer Information Systems and Organization Structures"; Communications of the ACM, Aug. 1981, Volume 24 #8, pg 679

185. "Computer System Simulation in Pascal"; Software-Practice and Experience, Aug. 1982, Volume 12, Issue #8, pg 777

254. "Concepts and Criteria to Assess Acceptability of Simulation Studies: A Frame of Reference"; Communications of the ACM; April 1981, Volume 24 #4, pg 180

322. "Contemporary Software Development Environment"; Communications of the ACM, Jan. 1982, Volume 25 #1, pg 318

141. "Contexts and Data Dependencies: A Synthesis"; IEEE Transactions on Pattern Analysis and Machine Intelligence, May 1983, Volume PAMI-5 #3, pg 237

167. "A Comparison of Programming Languages for Software Engineering"; Software-Practice and Experience, 1981, Volume 11, Issue-52, pg 3

170. "A Comparative Study of Task Communication in Ada"; Software-Practice and Experience, March 1981, Volume 11, Issue-3, pg 257

279. "A Computer Aid for the Analysis of Complex Systems"; The Computer Journal, March 1980, Volume 23 #2, pg 98

311. "A Contextual Analysis of Pascal Programs"; Software-Practice and Experience, Feb. 1982, Volume 12, Issue #2, pg 195

265. "Control Flow and Data Structure Documentation: Two Experiments"; Communications of the ACM, Jan. 1982, Volume 25 #1, pg 55

147. "Controlling the Complexity of Menu Networks"; Communications of the ACM, Volume 25 #7, pg 412

258. "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment"; Communications of the ACM, Aug. 1981, Volume 24 #8, pg 563

237. "Critical Event Modeling: A Step Beyond System Level Testing"; IEEE Software Proceedings, July 25-28, 1983, pg 207

250. "The Cue Project"; IEEE Softfair Proceedings, July 25-28, 1983, pg 383

362. Program Listings; Martin Marietta IR&D D16S Project Report; Deborah Sinay, August 1983

305. "Data Abstraction, Structured Programming, and the Practicing Programmer"; Software-Practice and Experience, July 1981, Volume 11, Issue #7, pg 607

195. "Data Processing in Blue Jeans"; Computer, March 1983, pg 66

37. "Decision Aids for Battle Management" (Battle Staff) Final Report Arthur D. Garmington (for USAF) Nov. 1982

41. "Decision Aids for Target Aggregation: Decision Situation Characteristics"; Michael L. Donnell April 1982

40. "Decision Aids for Target Aggregation: Decision Situation Characteristics Appendices"; Michael L. Donnell May 1982

39. "Decision Aids for Target Aggregation: Technology Review and Decision and Selection"; A. Joseph Rockmore May 1982

38. "Decision Analysis and Artificial Intelligence: Applications to Senior Battlestaff Decisions"; Paul E. Lehner (for USAF) Sept. 1981

223. "Decision Tables"; Software-Practice and Experience, 1983, Volume 13, pg 523

179. "Decomposition of Flowchart Schemata"; The Computer Journal, Aug 1981, Volume 24 #3, pg 258

87. "Definition of the Command Language Interface in the Tactical Fusion Center (TFC)"; OASIS program document

46. "Dimensions of Representation"; Daniel G. Bobrow Xerox, Palo Alto Research Center July 1975

313. "Description of a Menu Creation and Interpretation System"; Software-Practice and Experience, March 1982, Volume 12, Issue #3, pg 269

301. "A Design Language for the Definition of a Retrieval System Interface for Casual Users of a Relational Database"; Software-Practice and Experience, May 1981, Volume 11, Issue #5, pg 521

318. "A Design Medium for Software"; Software-Practice and Experience, June 1980, Volume 12, Issue #6, pg 497

115. "The Design of a Family of Applications-Oriented Requirements Languages"; Alan M. Davis (from W. Rzepka)

62. "Design of a Separable Transition-Diagram Compiler"; Melvin E. Conway July 1963 (Communications of the ACM)

343. "Design Principles Behind Smalltalk"; Daniel H. H. Ingalls, Smalltalk-80 System; Byte Magazine August 1981

206. "Design Rules Based on Analyses of Human Error"; Communications of the ACM, Volume 26 #4, pg 254

44. "Design Specification Validation," Final Technical Report RADC June 1981

221. "A Device-Independent Network Graphics System"; Computer Graphics, July 1983, Volume 17 #3, pg 167

108. "Devising a Laboratory to Simulate C^3I Operations"; Douglas B. Dahnn

187. "Development Methodologies for Scientific Software"; Software-Practice and Experience, Dec. 1982, Volume 12, Issue #12, pg 1085

92. "The Development of an Intelligent, Trainable Graphic Display Assistant for the Decisionmaker"; A. Morse, R. Kohler, and M. Sutherlant July 1982

264. "Diagram: A Grammar for Dialogues"; Communications of the ACM, Jan. 1982, Volume 25 #1, pg 27

134. "A Diagrammatic Notation for Abstract Syntax and Abstract Structured Objects"; IEEE Transactions on Software Engineering, May 1983, Volume SE-9 #3, pg 280

168. "Dialog: A Schema for the Quick and Effective Production of Interactive Applications Software"; Software-Practice and Experience, March 1981, Volume 11, Issue-3, pg 205

183. "A Dialogue Generator"; Software-Practice and Experience, Aug. 1982, volume 12, Issue #8, pg 693

94. "A Dialogue Simulation Tool for Use in the Design of Interactive Computer Systems"; D.R. Lenorovitz and H.R. Ramsey

196. "Direct Manipulation: A Step Beyond Programming Languages"; Computer, August 1983, pg 57

290. "Direct Implementations of Algebraic Specification of Abstract Data-Types"; IEEE Transactions on Software Engineering, Jan. 1982, Volume SE-8 #5, pg 12

102. "Display Applications in Command, Control and Communications Systems"; D.N. Grover and D.R. Lenorovitz

184. "The Distributed Programming Language SR-Mechanisms, Design and Implementation"; Software-Practice and Experience, Aug. 1982, Volume 12, Issue #8, pg 719

77. "Distributed Software Engineering Control Process," Volume I, Technical Proposal; Martin Marietta February 1983

78. "Distributed Software Engineering Control Process," Volume II (Volume I - See #77), Management and Resource Plan; Martin Marietta February 1983

260. "Documentation for Model: A Hierarchical Approach"; Communications of the ACM, Aug. 1981, Volume 24 #8, pg 728

32. DOD 7935.1-S Automated Data Systems Documentation Standards 13 Sept. 1977

306. "Dynamic Program Building"; Software-Practice and Experience, Aug. 1981, Volume 11, Issue #8, pg 853

207. "The Dynamics of Software Project Scheduling"; Communications of the ACM, May 1983, Volume 26 #5, pg 340

252. "The Effect of Programming Team Structures on Programming Task"; Communications of the ACM, Feb. 1981, Volume 24 #2, pg 106

203. "An Effective Graphic Vocabulary"; IEEE Computer Graphics and Applications, March/April 1983, pg 46

251. "The Emperor's Old Clothes"; Communications of the ACM, Feb. 1981, Volume 24 #2, pg 75

55. "EP-2: An Exemplary Programming System," Rand; W.S. Faught Feb. 1980

47. "Empirical Estimates of Program Entropy"; Richard E. Sweet Xerox, Palo Alto Research Center Sept. 1978

82. "Empirical and Formal Language Design Applied to a Unified Control Construct for Interactive Computing"; David W. Embley Nov. 1977

113. "Enhancement of System Design and Simulation Via General System Theories"; J. Talavage (1981 Winter Simulation Conference Proceedings)

175. "Entity Life Cycle Models and Their Applicability to Information Systems Development Life Cycles: A Framework for Information Systems Design and Implementation"; The Computer Journal, Aug. 1982, Volume 25 #3, pg 307

204. "Error Messages: The Neglected Area of the Man/Machine Interface?"; Communications of the ACM, April 1983, Volume 26 #4, pg 246

57. "The Evolution of Cognitive Structures and Process"; Barbara Hayes-Roth October 1976

243. "The Evolutionary Approach to Building the Joseph Software Development Environment"; IEEE Softfair Proceedings, July 25-28, 1983, pg 317

53. "Exemplary Programming in Rita"; D.A. Waterman, October 1977

239. "Experience With Tool-Kit Approach in SMEF Prototyping," IEEE Softfair Proceedings, July 25-28, 1983, pg 223

229. "Experiences With Smalltalk-80 For Application Development"; IEEE Softfair Proceedings, July 25-28, 1983, pg 61

261. "An Experiment Study of the Human/Computer Interface"; Communication of the ACM, Aug. 1981, Volume 24 #8, pg 752

328. "An Experimental Program Transformation and Synthesis System"; Artificial Intelligence, 1981, Volume 16, pg 1

281. "Extended Attribute Grammars"; The Computer Journal, May 1983, Volume 26 #2, pg 142

214. "Fifth-Generation Hardware Takes Shape"; Electronics, July 1983, pg 1

132. "File Structures, Program Structures, and Attributed Grammars"; IEEE Transactions on Software Engineering, May 1983, Volume SE-9 #3, pg 260

97. "Final Report of the GSPC State-of-the-Art Subcommittee"; Computer Graphics, June 1978, Volume 12 #1-2, pg 14

104. "Flowcharts Versus Program Design Languages: An Experimental Comparison"; H.R. Ramsey, M.E. Atwood, J.R. VanDoren, June 1983

59. "Formal Grammar and Human Factors Design of an Interactive Graphics System"; Phyllis Reisner; IEEE Transactions of Software Engineering, March 1981

154. "A Fortran Programming Methodology Based on Data Abstraction," Communications of the ACM, Volume 25 #7, pg 686

111. "Foundations for an Information Technology"; Tuncer I. Oren (1981 Winter Simulation Conference Proceedings)

309. "A Framework for Modeling Graphic Interactions"; Software-Practice and Experience, Feb 1982, Volume 12, Issue #2, pg 141

321. "The Future of Programming"; Communications of the ACM, Jan 1982, Volume 25 #1, pg 196

93. "General Technique for Communications Protocol Validation" C.H. West July 1978

139. "A Generalized Query-By-Example Data Manipulation Language Based on Data Logic"; IEEE Transactions on Software Engineering, January 1983, Volume SE-9 #1, pg 40

262. "A Generalized User Interface for Application Programs"; Communication of the ACM, Aug 1981, Volume 24 #8, pg 796

69. "GPM Technical Volume and User's Guide (General Processor Model)"; D. G. Glinos; Martin Marietta Aerospace Technical Report

95. "A Graph-Theoretic Language Extension for an Interactive Computer Graphics Environment"; James P. DelGrande May 1979

209. "Graphic Design for Computer Graphics"; IEEE Computer Graphics and Applications, July 1983, pg 63

106. "Grasp: A Software Development System Using D-Charts"; D.A. Workman December 1979, Software-Practice and Experience 1/83 W/105

181. "Hades - A Command Environment That Supports Structure"; Software-Practice and Experience, July 1982, Volume 12, Issue #7, pg 641

275. "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty"; ACM Computing Survey, March 1980, Volume 12 #1, pg 213

188. "Hierarchically Structured Production Rules"; The Computer Journal, Feb 1983, Volume 26 #1, pg 1

144. "Higher Level Programming and Data Abstractions - A Case Study Using Enhanced C"; Software-Practice and Experience, 1983, Volume 13, pg 577

155. "HISDL - A Structured Description Language"; Communications of the ACM, Volume 25 #7, pg 823

300. "How a Computer Should Talk to People"; IBM Systems Journal, 1982, Volume 21 #4, pg 424

298. "How Data Flow Can Improve Application Development Productivity"; IBM Systems Journal, 1982, Volume 21 #2, pg 162

266. "A Human/Computer Interface to Accommodate User Learning Stages"; Communications of the ACM, Jan 1982, Volume 25 #1, pg 100

152. "A Human Factors Study of Color Notation Systems for Computer Graphics"; Communications of the ACM, Volume 25 #7, pg 547

190. "Human Performance in Interactive Graphics Operations"; The Computer Journal, Feb 1983, Volume 26 #1, pg 93

205. "The Humanization of Computer Interfaces"; Communications of the ACM, Volume 26 #4, pg 252

201. "Iconic Interfacing," IEEE Computer Graphics and Applications, Applications, March/April 1983, pg 8

114. "Impact of General Systems Orientation: Present and Future" Bernard P. Zeigler (1981 Winter Simulation Conference Proceedings)

49. "The Impact of Rapid Prototyping on Specifying User Requirements-Rapid Prototyping Continued"; ACM Sigsoft, April 1982

56. "Implications of Human Pattern Processing for the Design of Artificial Knowledge Systems," Barbara Hayes-Roth April 1977

28. "Improved Approach to Procuring and Developing Enhancements to a Baseline System Under the Evolutionary System Development Approach"; Harry Kottcamp (W/21), Martin Marietta Aerospace memo

150. "Improving Computer Program Readability to Aid Modification"; Communications of the ACM, Volume 25 #7, pg 512

220. "Incense: A System for Displaying Data Structures"; Computer Graphics, July 1983, Volume 17 #3, pg 115

178. "Increasing Computer System Productivity Software and Hardware Methods: A Comparative Study"; The Computer Journal, Aug 1981, Volume 24 #3,
pg 210

236. "The Index Development Environment Workbench"; IEEE Softfair Proceedings, July 25-28, 1983, pg 200

189. "In Favour of System Prototypes and Their Integration Into the Systems Development Cycle"; The Computer Journal, Feb 1983, volume 26 #1, pg 36

131. "Input-Output Tools: A Language Facility for Interactive and Real-Time Systems"; IEEE Transactions on Software Engineering, May 1983, Volume SE-9 #3, pg 247

241. "An Integrated Interactive Design Environment for Taxis"; IEEE Softfair Proceedings, July 25-28, 1983, pg 298

356. "The Integrated Requirements Implementation System (IRIS) as Applied to C3 Systems," May 1983 (Larry Trometer) Martin Marietta Aerospace Technical Report

50. "An Integrated Set of Tools to Automate the Software Life Cycle"; GTE Labs

64. "An Integrated System Analysis and Engineering (SAE) Toolkit," R. Newman May 26, 1983 Martin Marietta Aerospace Technical Report

96. "An Interactive System for the Construction Animation of Systems Dynamics Models"; J.P. DelGrande and L. Mezei January 1979

240. "An Introduction to Editor Allan Poe"; IEEE Softfair Proceedings, July 25-28, 1983, pg 245

336. "Introducing the Smalltalk-80 System"; Adele Goldberg Smalltalk-80 System--Byte Magazine August 1981

143. "Introduction to Enhanced C (EC)"; Software-Practice and Experience, 1983, Volume 13, pg 551

159. "An Investigation of Computer Coaching for Informal Learning Activities"; International Journal of Man-Machine Studies, Jan 1979, Volume 11 #1, pg 5

357. IRIS D-29R 1983, Ronald A. Bena, Martin Marietta Technical Report

355. IRIS Design Concept Document Project D-29R March 1983, Larry Trometer, Martin Marietta Technical Report

307. "Is Block Structure Necessary?"; Software-Practice and Experience, Aug. 1981, Volume 11, Issue #8, pg 853

346. "Is the Smalltalk-80 System for Children?"; Adele Goldberg and Joan Ross, Smalltalk-80 System--Byte Magazine August 1981

65. "Language Development Tools on the Unix System," Stephen C. Johnson (Bell Laboratories) August 1980

137. "Language Features for Access Control"; IEEE Transactions on Software Engineering, January 1983, Volume SE-9 #1, pg 16

271. "Leave and Recall: Primitives for Procedural Programming"; Software Practical and Experience, 1980, Volume 10, pg 127

89. "LR Parsing"; A.V. Aho and S. C. Johnson, June 1974

79. "Medl-D User's Guide," Martin Marietta Technical Report

80. "Medl-R User's Guide," Martin Marietta Technical Report

5. MIL-S-52779 S/W Q.A. Program Requirements 05 April 1964

3. MIL-STD-483 (USAF) Configuration Management Practices 21 March 1979

1. MIL-STD-490 Specification Practices 30 October 1968

4. MIL-STD-499A Military Standards Engineering Management 01 May 1974

2. MIL-STD-1679 (Navy) Weapon System S/W Development 01 Dec 1978

6. MIL-Q-9858A Quality Program Requirements 16 Dec 1963

297. "Macro Implementation of a Structured Assembly Language"; IEEE Transactions on Software Engineering, May 1980, Volume SE-8 #3, pg 284

228. "The Message/Object Programming Model"; IEEE Softfair Proceedings, July 25-28, 1983, pg 51

60. "Military Message Systems: Current Status and Future Directions"; Constance L. Heitmeyer and Stanley H. Wilson Sept. 1980 IEEE Transactions on Communications

222. "Minimal GKS"; Computer Graphics, July 1983, Volume 17 #3, pg 183

165. "MM/1, A Man-Machine Interface"; Software-Practice and Experience, 1980, Volume 10, pg 751

319. "Modeling and Validating the Man-Machine Interface"; Software-Practice and Experience, June 1980, Volume 12, Issue #6, pg 557

169. "A Modula Based Language Supporting Hierarchical Development and Verification"; Software-Practice and Experience, March 1981, Volume 11, Issue-3, pg 237

317. "A Multi-User Operating System for Transaction Processing, Written in Concurrent Pascal"; Software-Practice and Experience; May 1982, Volume 12, Issue #5, pg 445

67. "Multiparty Grammar and Related Features for Defining Interactive Systems"; Ben Shneiderman April 1982

120. "Mumps Language Standard;" Mumps Development Committee System Sept. 1977

353. "Navy Space Sensors Face Tough Requirements"; Military Electronics, April 1983, pg 37

135. "An Object-Oriented Command Language"; IEEE Transaction on Software Engineering, January 1983, Volume SE-9 #1, pg 1

253. "On Approaches to the Study of Social Issues in Computing"; Communications of the ACM, Feb. 1981, Volume 24 #2, pg 146

162. "On Generation of Inductive Hypotheses"; International Journal of Man-Machine Studies, July 1977, Volume 9 #4, pg 415

149. "On the Inevitable Intertwining of Specification and Implementation"; Communications of the ACM, Volume 25 #7, pg 438

13. "On the Management of USAFE C3 System Acquisition"; Lieutenant Col. Frank M. Hernon, 10 June 1982, Technical Note

224. "On the Realization of Extended Control Structure in Fortran"; Software-Practice and Experience, 1983, volume 13, pg 431

68. "On the Use of Transition Diagrams in the Design of a User Interface for an Interactive Computer System, David L. Parnas 1969

295. "An Operational Approach to Requirements Specification for Embedded Systems"; IEEE Transactions on Software Engineering, May 1980, Volume SE-8 #3, pg 250

42. "An Overview of Computer-Based Natural Language Processing"; William B. Gevarter, April 1983

43. "An Overview of Expert Systems"; William B. Gevarter, May 1982

230. "Ovide: A Software Package for Application Development"; IEEE Softfair Proceedings, July 25-28, 1983, pg 61

210. "P-System Infiltrates Multiuser World"; Electronics, Feb. 1983, pg 75

14. "P3I in the C^3I Community - A Model and Implementation"; James W. Youngberg, Major USAF

103. "PDL-Program Design Language Reference Guide," February 1977

359. "Penetration Analysis Subsystem Data Sensitivity Analysis" - Final Report, Sept. 1980, RADC Technical Report

358. "Penetration Evaluation Model Users Guide"; Martin Marietta Aerospace

288. "Perceptual Components of Computer Displays"; IEEE Computer Graphics and Applications, May 1982, Volume 2 #3, pg 23

142. "Planning in Time: Windows and Durations for Activities and Goals"; Pattern Analysis and Machine Intelligence, May 1983, Volume PAMI-5 #3, pg 246

235. "Platine: A Software Engineering Environment"; IEEE Softfair Proceedings, July 25-28, 1983, pg 193

247. "Pride-Automated System Design Methodology"; IEEE Softfair Proceedings, July 25-28, 1983, pg 351

101. "A Primer on Relational Data Base Concepts," G. Sandbery, 1981

148. "Principles of Package Design"; Communications of the ACM, Volume 25 #7, pg 419

276. "Probabilistic Languages: A Review and Some Open Questions"; Computing Surveys, March 1980, Volume 12 #1, pg 361

48. Proceedings of the Workshop on Data Abstractions, Databases and Conceptual Modeling; ACM, June 1980

74. Proceeding SCE/ISDOS User's Workshop, ISDOS Project October 1978

164. "A Process Oriented Simulation Model Specification and Documentation Language"; Software-Practice and Experience, 1980, Volume 10, pg 721

197. "Program Development"; Communications of the ACM, Jan. 1983, Volume 26 #1, pg 70

293. "Program Specification Applied to a Text Formatter"; IEEE Transactions on Software Engineering, Sept. 1982, Volume SE-8 #5, pg 490

153. "A Program Testing Assistant"; Communications of the ACM, Volume 25 #7, pg 625

191. "The Programmable Compiler"; Computer, March 1983, pg 35

289. "The Programmer's Apprentice; Knowledge Based Program Editing"; IEEE Transactions on Software Engineering, Jan. 1982, Volume SE-8 #1, pgl

283. "Programmer-Defined Control Operations"; The Computer Journal, May 1983, Volume 26 #2, pg 175

282. "Programming Denotational Semantics"; The Computer Journal, May 1983, Volume 26 #2, pg 164

174. "The Programming Language BPL"; The Computer Journal, August 1982, Volume 25 #3, pg 289

138. "Programming Language Constructs for Screen Definition"; IEEE Transactions on Software Engineering, January 1983, Volume SE-9 #1, pg 31

352. "Protecting Stored Data Remains a Serious Problem (Computer Security)"; Military Electronics, April 1983, pg 26

238. "Proto-Cycling: A New Method for Application Development Using Fourth Generation Languages"; IEEE Softfair Proceedings, July 25-28, 1983, pg 217

119. "A Prototyping and Simulation Approach to Interactive Computer System Design"; Paul R. Hanau and David R. Lenorovitz

118. "Prototyping and Simulation Tools for User/Computer Dialogue Design"; Paul R. Hanau and David R. Lenorovitz

208. "Prototyping Interactive Information Systems," Communications of the ACM, Volume 26 #4, pg 347

70. "PSL/PSA A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems"; ISDOS Project, Daniel Teichrowew and E.A. Hershey III August 1976

98. "Q Charts - A Method of Specification"; G. Duncan March 1979

15. RADC Computer S/W Development Specification - General Specifications 30 June 1977

284. "RCC - A User-Extensible Systems Implementation Language"; The Computer Journal, August 1980, Volume 23 #3, pg 213

151. "Relating Sentences and Semantic Networks With Procedural Logic"; Communications of the ACM, Volume 25 #7

267. "Relational Database: A Practical Foundation for Productivity"; Communications of the ACM, Jan 1982, Volume 25 #1, pg 109981,

45. "Repair Theory: A Generative Theory of Bugs in Procedural Skill"; Xerox Palo Alto Research Center Aug. 1980

270. "A Review and Evaluation of Software Science"; Computing Surveys, March 1978, Volume 10 #1, pg 3

72. Revs Users Manual (SREP Final Report, Volume II) By: M.E. Dyer August 1977

117. "Research Directions and Discussions" (Research Direction in Software Technology) 1980 By: B. H. Luskov and V. Berzins

302. "Scenarios: An Event Management Package"; Software-Practice and Experience, May 1981, Volume 11, Issue #5, pg 521

249. "SEA/I-Application Software Productivity System"; IEEE Softfair Proceedings, July 25-28, 1983, pg 375

157. "Self-Assessment Procedure X"; Communications of the ACM, Volume 25 #7, pg 883

136. "Simulation of Procedure Variables Using Ada Task"; IEEE Transactions on Software Engineering, January 1983, Volume SE-9 #1, pg 13

172. "Simulation Tools in Computer System Design Methodologies"; The Computer Journal, Feb. 1981, Volume 24 #1, pg 25

140. "Skills, Rules, Knowledge; Signals, Signs, and Symbols, and Other Distinctions in Human Performance Models"; IEEE Transactions on Software Engineering, May/June 1983, Volume SMC-13 #3, pg 257

337. "The Smalltalk-80 System," Xerox Learning Research Group Smalltalk-80 System--Byte Magazine August 1981

338. "Object-Oriented Software Systems", David Robson, Smalltalk-80 System--Byte Magazine August 1981

268. "The Organization of the Living: A Theory of the Living Organizations"; International Journal of Man-Machine Studies, May 1975, Volume 7 #3, pg 313

339. "The Smalltalk Environment"; Larry Tesler, Smalltalk-80 System--Byte Magazine August 1981

341. "The Smalltalk Graphics Kernel"; Daniel H. H. Ingalls Smalltalk-80 System--Byte Magazine August 1981

344. "The Smalltalk-80 Virtual Machine"; Glenn Krasner, Smalltalk-80 System--Byte Magazine August 1981

110. "Snapshot Thoughts on Rapid Prototyping," Howie Dahl, July 1983, Martin Marietta Aerospace Technical Note

274. "Social Analyses of Computing: Theoretical Perspectives in Recent Empirical Research"; ACM Computing Surveys, March 1980, Volume 12 #1, pg 61

278. "Social Aspects of Systems Analysis"; The Computer Journal, Feb. 1980, Volume 23 #1, pg 5

88. "The Software Designer Workbench (DWB)"; Paul A. Scheffer, Martin Marietta Technical Report

303. "Software Engineering: An Example of Misuse"; Software-Practice and Experience, June 1981, Volume 11, Issue #6, pg 629

156. "Software Engineering for the Cobol Environment"; Communications of the ACM, Volume 25 #7, pg 574

314. "The Software Engineering of a Micro Computer Application System"; Software-Practice and Experience, April 1982, Volume 12, Issue #4, pg 309

245. "A Software Development Database"; IEEE Softfair Proceedings, July 25-28, 1983, pg 337

29. "Software Development Methodology," Al Florence, 3 May 1983, Martin Marietta Aerospace Technical Report

232. "Software Must Move! A Description of the Software Assembly Lines"; IEEE Softfair Proceedings, July 25-28, 1983, pg 119

213. "Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support"; IEEE Transactions on Software Engineering, March 1983, Volume SE-9 #2, pg 155

246. "Software Tools Archive (STAR)"; IEEE Softfair Proceedings, July 25-28, 1983, pg 343

91. "Software Quality Attribute Definitions," Martin Marietta Technical Report

194. "Software Quality Improvement"; Computer, March 1983, pg 66

263. "Some Practical Experience With a Software Quality Assurance Program"; Communications of the ACM, Jan. 1982, Volume 25 #1, pg 4

161. "Sophie: A Step Toward Creating a Reactive Learning Environment"; International Journal of Man-Machine Studies, Sept. 1975, Volume 7 #5, pg 675

51. Special Issue on Rapid Prototyping; ACM Sigsoft, April 1982

292. "Specification and Verification of Communication Protocols in Affirm Using State of Transition Models"; IEEE Transactions on Software Engineering, Sept. 1982, Volume SE-8 #5, pg 460

86. Specification Languages: RSL, MSL, DSL, and ESL

291. "A Specification Method for Specifying Data and Procedural Abstractions"; IEEE Transactions on Software Engineering, Sept. 1982, Volume SE-8 #5, pg 449

294. "Specification of Forms Processing and Business Procedures for Office Automation"; IEEE Transactions of Software Engineering, Sept. 1982, Volume SE-8 #5, pg 499

146. "A Specification Schema for Indenting Programs"; Software-Practice and Experience, 1983, Volume 13, pg 163

316. "Specifications: Formal and Informal-A Case Study"; Software-Practice and Experience, May 1982, Volume 12, Issue #5, pg 433

27. "SREM Evaluation - Final Report" - Volume 1 (Draft), Martin Marietta Aerospace, A. Stone, D. Hartschuh, B. Castor April 1983

73. "SREM: Requirements Development Using SREM Technology," Volume II, TRW, June 1979

76. SREM-Software Requirements Engineering Methodology," TRW, TRW Class October 1977 Volume II

105. "Stoic, An Interactive Programming System for Dedicated Computing," J.M. Sachs and S.S. Burns, December 1980, Software-Practice and Experience 1/83 W/106

272. "Strategies For Information Requirements Determination"; Systems Journal, 1982, Volume 21 #1, pg 4

176. "Structured System Analysis and Design Using Standard Flowcharting Symbols"; The Computer Journal, Nov. 1981, Volume 24 #4, pg 295

287. "A Style for Writing the Syntactic Portions of Complete Definitions of Programming Languages"; The Computer Journal, May 1981, Volume 24 #2, pg 143

242. "Super PDL-A Software Design Tool"; IEEE Softfair Proceedings, July 25-28, 1983, pg 307

349. "Superposition Provides an Intelligence Fusion"; Military Electronics, April 1983

350. "Surveillance is the Key Soviet Space Mission"; Military Electronics, April 1983, pg 18

163. "A Survey of Information Requirements Analysis"; Computing Surveys, Dec. 1977, Volume 9 #4, pg 273

192. "Strategy for a DOD Software Initiative"; Computer, March 1983, pg 52

218. "Syngraph: A Graphic User Interface Generator"; Computer Graphics, July 1983, Volume 17 #3, pg 43

308. "The Syntax of Interactive Command Languages: A Framework for Design"; Software-Practice and Experience, Jan. 1982, Volume 12, Issue #2, pg 141

280. "System Conventions for Nonprocedural Languages"; The Computer Journal, March 1980, Volume 23 #2, pg 132

256. "System Design for Usability"; Communications of the ACM, Aug. 1981, Volume 24 #8, pg 494

22. "System Acquisition," Technical Report, AFCEA, for General Marsh

223. "System Operational Design for the TFC Users Interface Phase II (TUI-II)," Oasis Program Specification

35. "Systems Software Support for the USAFE TFC," OASIS Specification

173. "Systematics: Its Syntax and Semantics as a Query Language (1)"; The Computer Journal, Feb. 1981, Volume 24 #1, pg 56

286. "Systematics: Its Syntax and Semantics as a Query Language (2)"; The Computer Journal, May 1981, Volume 24 #2, pg 125

171. "A Taxonomy of Current Approaches to Systems Analysis"; The Computer Journal, Volume 25 #1, Feb. 1982

198. "A Technique for Software Module Specification With Example"; Communications of the ACM, Jan. 1983, Volume 26 #1, pg 75

299. "Technique for Assessing External Design of Software"; IBM Systems Journal, 1982, Volume 21 #2, pg 211

25. "TFC-User Interface, Phase II Man-Machine Interface (MMI) Guidelines," 15 Oct. 1982 Oasis Program

255. "The Time and State Relationships in Simulation Modeling"; Communications of the ACM, Feb. 1981, Volume 24 #2, pg 173

244. "The Toolpack/1st Programming Environment"; IEEE Softfair Proceedings, July 25-28, 1983, pg 326

312. "A Tool to Aid in the Installation of Complex Software Systems"; Software-Practice and Experience, March 1982, Volume 12, Issue #3, pg 251

231. "Tools and Methodologies: The Perfect Match or the Odd Couple"; IEEE Softfair Proceedings, July 25-28, 1983, pg 95

347. "Toolbox: A Smalltalk Illustration System"; William Bowman and Bob Flegal, Smalltalk-80 System--Byte Magazine August 1981

273. "Towards an Integrated Development Environment"; Systems Journal, Volume 21 #1, pg 81

217. "Towards a Comprehensive User Interface Management System"; Computer Graphics, July 1983, Volume 17 #3, pg 35

158. "Towards a Theory of the Cognitive Processes in Computer Programming"; International Journal of Man-Machine Studies, Nov. 1977, Volume 9 #6, pg 737

277. "Towards Comprehensive Specification"; The Computer Journal, Aug. 1979, Volume 22 #3, pg 195

61. "Transition Network Grammars for Natural Language Analysis"; W.A. Woods (Computational Linguistics)

84. "Translation of Decision Tables," Udo W. Pooch

360. "TSD Methodology Technical Report," Draft Final Report; Martin Marietta Aerospace

225. "Tutorial: Data Structure, Types, and Abstraction"; Computer, April 1980, Volume 13, pg 67

296. "Understanding and Documenting Programs"; IEEE Transactions on Software Engineering, May 1980, Volume SE-8 #3, pg 270

234. "The Unified Support Environment: Tool Support for the User Software Engineering Methodology"; IEEE Softfair Proceedings, July 25-28, 1983, pg 145

182. "A Unified Theory for Software Production"; Software-Practice and Experience, Volume 12, Issue #7, pg 683

202. "The Use of a Sophisticated Graphic's Interface in Computer-Assisted Instruction"; IEEE Computer Graphics and Applications, March/April 1983, pg 25

24. "The Use of Prototype MMI to Resolve Requirement Issues With C2I Users"

145. "User Acceptance: Design Considerations for a Program Generator"; Software-Practice and Experience, 1983, Volume 13, pg 101

257. "A User-Friendly Algorithm"; Communications of the ACM, Aug. 1981, Volume 24 #8, pg 556

340. "User-Oriented Descriptions of Smalltalk Systems"; Trygve M.H. Reenskaug, Smalltalk-80 System--Byte Magazine August 1981

310. "Uses of the Simula Process Concept"; Software - Practice and Experience, Feb. 1982, Volume 12, Issue #2, pg 153

63. "Using Formal Specifications in the Design of Human-Computer Interface"; Robert J. K. Jacob April 1963 (Communications of the ACM)

177. "Validation of an Analytic Model of Computer Performance"; The Computer Journal, Nov. 1981, Volume 24 #4, pg 347

83. "The Vienna Definition Language," Peter Wegner, Computer Surveys

348. "Virtual Memory for an Object-Oriented Language"; Ted Kaehler Smalltalk-80 System--Byte Magazine August 1981

233. "What About CAD/CAM for Software? The Argus Concept"; IEEE Softfair Proceedings, July 25-28, 1983, pg 129

PRELIMINARY RESULTS OF REVIEW OF AFR 800.14

PARA NO.	TOPIC	ISSUE	SUGGESTED CHANGE
3.1.6	level of simulation	relate prototyping to simulation	"...the level of simulation and prototyping ..."
vol.1, attach 1	def. of simulation	include prototyping definition or add a new definition	"11. Simulation or prototyping..."
2-3.a	conceptual phase activities	include prototyping as an activity	"...tradeoffs, studies, prototyping, and analyses..."
2-5.	Full scale development phase definition	a single development model is not necessarily optimal	add "During this phase a prototype or sequence of prototypes may be built to aid creation of the system."
2-5.a	" " " "	Specifications are stated as the only basis for PDR for a CP	"...and become, in addition to the results of prototyping activity, the basis for the PDR of the computer program."
2-8	Computer Program development in the System Acquisition Life Cycle	introduce prototyping as a structured feedback mechanism	add data to the paragraph about prototyping as a structured feedback mechanism. Contents TBD.
2-8	Figure 2-1	the life cycle for a computer program	add figure elements corresponding to TBD para above.

PRELIMINARY RESULTS OF REVIEW OF AFR 800.14

PARA NO.	TOPIC	ISSUE	SUGGESTED CHANGE
2-8.a	activities of analysis	include prototyping as an activity	"...analyses, tradeoff studies and prototyping performed..."
2-8.b	activities of the design phase	supplement goals of design phase with prototype production	"..., detail flow charts, and working prototypes when appropriate..."
2-8.b	" " "	"	fix above now requires a TBD para. in a TBD location defining where appropriate. Ref DODS 5000.2
2-8.b	" " "	"	last sentence of para may need change concerning review process and prototyping.
3-3	Development of computer resource requirements	This paragraph focuses on paper based studies alone	include a TBD sentence in a TBD location to make prototyping a viable basis for establishment of computer resource requirements.
3-6.b	Program Mgmt Directive Life Cycle	use of prototyping to support functional analyses	"Modeling, simulation, and prototyping..."

PRELIMINARY RESULTS OF REVIEW OF AFR 800.14

PARA NO.	TOPIC	ISSUE	SUGGESTED CHANGE
3-7.k	Program Mgmt Plan	prototyping facility support for CP support	"...for simulation, prototyping, integration, and other..."
3-9.o	Computer Program Development Plan	include planning for prototyping	"Simulation techniques and tasks and prototyping techniques and tasks."
4-5.c	System Engineering Process	include prototyping as a support function in the System Engineering process	Append "At each level validation may be supported by simulations, prototypes and other techniques."
4-9.a	Formal Technical Reviews	include results of prototyping activities	include "(6) The results of any prototyping activities."
4-9.b	"	basis for System Design Reviews	include "(8) The results of any prototyping activities."
4-9.c	"	basis for Preliminary Design Reviews	include "(7) The results of any prototyping activities."
4-9.d	"	basis for Critical Design Reviews	include (5) "...detailed flow charts and results from any prototyping activities ..."
5-6	Computer Program V&V	This paragraph presumes some degree of prototyping	TBD

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END

DTIC

6-86