

AD-A259 232



2

NPS-AS-93-005

NAVAL POSTGRADUATE SCHOOL

Monterey, California



DTIC
ELECTE
JAN 6 1993
S C D

**Prototyping with Application Generators:
Lessons Learned from the
Naval Aviation Logistics Command
Management Information System Case**

**Tung Bui
Cheryl D. Blake
James C. Emery**

October 1992

93-00345



5328

Approved for public release; distribution is unlimited.

Prepared for: Director of Information, OASI (C3I),
Washington, D.C. 20301-3040

93 1 05 051

NAVAL POSTGRADUATE SCHOOL
Monterey, California

RADM R. W. West, Jr.
Superintendent

Harrison Shull
Provost

The report was prepared for the Director of Defense Information, OASI (C3I), Washington, D.C.. This research was funded by DoD Washington Headquarters Services, IAD, The Pentagon, Washington, D.C.

Reproduction of all or part of this report is authorized.

This report was prepared by:



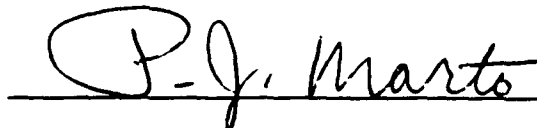
Tung X. Bui
Dept of Admin. Sciences

Reviewed by:



David R. Whipple, Chairman
Department of Administrative Sciences

Released by:



Paul J. Marto, Dean of Research

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management Budget, Paper Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE October 1992		3. REPORT TYPE AND DATES COVERED Technical Report, 1992	
4. TITLE AND SUBTITLE Prototyping with Application Generators: Lessons Learned from the Naval Aviation Logistics Command Management information System Case				5. FUNDING NUMBERS MIPR DXAM 20001	
6. AUTHOR(S) Tung X. Bui, Cheryl D. Blake and James C. Emery					
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Administrative Sciences Department Naval Postgraduate School Monterey, Ca 93943				8. PERFORMING ORGANIZATION REPORT NUMBER NPS-AS-93-005	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Director of Information OASI (C3I) Washington, D.C. 20301-3040				10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES Prototyping and Incremental Development, In-house Development, Application Generators, Use of					
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) Prototyping with Application Generators: Lessons Learned from the Naval Aviation Logistics Command Management Information System Case. This study reports the lessons learned from the use of software application generators by the Navy Management Systems Support Office (NAVMASSO) to develop a management information system to automate manual Naval aviation maintenance tasks-NALCOMIS. With the use of a fourth-generation programming language, NAVMASSO has been able to salvage an information system project that was projecting cost and schedule overrun and experiencing management difficulties with outside developers.					
14. SUBJECT TERMS				15. NUMBER OF PAGES 65	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited		



**CASE STUDY SERIES
ON IMPLEMENTATION PRACTICES OF
MANAGEMENT INFORMATION SYSTEMS
IN THE DEPARTMENT OF DEFENSE**

**Prototyping with Application Generators:
Lessons Learned from the
Naval Aviation Logistics Command
Management Information System Case**

**Tung X. Bui
Cheryl D. Blake
James C. Emery**

DTIC QUALITY INSPECTED 5

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

**Department of Administrative Sciences
Information Technology Management Curriculum
Monterey, California**

October 1992

Acknowledgments

The authors would like to thank Mr. Paul Strassmann, Director of Defense Information, OASD (C3I), for sponsoring the Case Study Series on effective use of information technology in DoD. We greatly appreciate the Naval Air Systems Command PMA-270 personnel for their support of this case study. Heartfelt thanks to the NALCOMIS Program Office, Naval Aviation Atlantic, and Navy Management Systems Support Office personnel for their unending patience, support, and encouragement, particularly CAPT P. T. Smiley, USN, LCDR Galen Ledeboer, USN, LCDR Ron Allen, USN, and AVCM Steve Shutler, USN.

We would like to thank Professor Sterling Sessions for sharing his interview techniques, Professor Bala Ramesh for his comments, LT Christine Donohue, USN, and LT MaryJo Elliott, USN, for their assistance with the case study format, and the officer-students of the Computer Systems Management class of September 1992 at the Naval Postgraduate School for their moral support.

Table of Contents

I.	Executive Summary	1
II.	Prototyping as a Development Alternative	7
	A. Classical Development Methodology	7
	B. Prototyping	9
	C. Fourth Generation Languages	12
	D. Evaluation Criteria	14
	E. Summary	16
III.	Transition to Prototyping: The NALCOMIS Case	17
	A. Mission	17
	B. Key Organizational Players	18
	1. Naval Air Systems Command PMA-270	18
	2. Navy Management Systems Support Office	18
	3. Type Commanders	19
	4. Fleet Design Team	20
	5. Fleet Design Review Group	20
	6. Commander Operational Test & Evaluation Force	20
	C. Background	20
	1. NALCOMIS/I	20
	2. NALCOMIS/II	21
	D. Functional Requirements of NALCOMIS/III	22

E.	Initial Development Strategy of NALCOMIS/III	24
1.	Difficulties Encountered with the Classical Development Approach	24
2.	First Development Experience with NALCOMIS/III	26
3.	Transition to Prototyping	27
IV.	NALCOMIS/III: Prototyping with an Application Generator	31
A.	Hardware Environment	31
1.	Prototype Hardware	31
2.	Operational Hardware	32
B.	The Development Process	32
C.	An Assessment of the NALCOMIS/III Prototyping Approach	34
1.	Software Development Status	34
2.	Development Costs	36
3.	Schedule	36
4.	Testing and Evaluation	37
5.	Training Adequacy	39
a.	Programmers' Training	39
b.	Users' Training	39
6.	Management Effectiveness	40
7.	Benefits Analysis	40
V.	Lessons Learned	43
A.	Dedication of Managers, Developers and Users is Crucial	43
B.	Prototyping Enables Systems to Exceed "Pre-Defined" Functional Requirements	43
C.	Prototyping Allows Rapid Recovery from Faulty Software Engineering Practices	44
D.	Existing Operational Test and Evaluation Methodology is Inappropriate for Evolutionary Development	44
E.	Design Documentation Should be Updated to Reflect Evolving System Design	45

F.	Management Must Provide a Proper Environment for Prototyping	45
G.	Application Generators Must be Carefully Selected	46
H.	Software Development Contract Characteristics Should be Reevaluated	46
I.	Current DoD Hardware Acquisition Regulations Hinder System Development	47
	Appendix	49
	Glossary of Terms	51
	References	53
	Bibliography	55

I. Executive Summary

Solving the Software Crisis

During the last decade, organizations have witnessed a phenomenon now known as the software crisis. Software costs have spiraled dramatically, becoming the largest item in information technology. Delivery schedules are seldom kept. As software grows rapidly to meet more complex requirements, quality has become a non-trivial issue.

Department of Defense (DoD) organizations are not immune to this crisis. In fact, in the wake of a declining budget and personnel reductions, there has been increased pressure on DoD to develop innovative ways to solve their software problems. New software engineering techniques are being explored to build information systems quickly, correctly, and cost-effectively.

This exploration is critical since current DoD standards and policy have so far favored systems to be built using a development methodology traditionally known as the "waterfall" model. This methodology is systematic and sequential, with the output of one phase acting as the input to the next. Although successful in some well-defined, highly structured, large-scale projects, its inflexibility has been one of the major sources of cost overruns, schedule slippage, and unsuitable end products.

Experience in the private sector has shown that prototyping is an alternative to the waterfall methodology. With prototyping, developers explore user requirements, experiment with ways to satisfy them, and enable the system design to evolve using a working model. Prototyping is known to be suitable for developing systems where

requirements are unclear, volatile, or cannot be communicated easily. Thanks to the constant feedback with users knowledgeable in the business area, prototyping has been successfully used to develop information systems that meet or exceed users requirements. Furthermore, the adoption of new development tools — such as fourth-generation languages (4GLs) — has increased software development productivity to the point that it is now possible to accomodate the frequent changes required in an evolving development process.

Shifting to Prototyping — The NALCOMIS Case

This case study reports the successful adoption of the prototyping approach by the Navy Management Systems Support Office (NAVMASSO). With the use of a 4GL, NAVMASSO has been able to salvage an information system project that was projecting cost and schedule overruns and experiencing management difficulties with outside developers.

The project involved is the Naval Aviation Logistics Command Management Information System (NALCOMIS) program. NALCOMIS was established by the Chief of Naval Operations (CNO) in 1975 to automate manual Naval aviation maintenance tasks. NALCOMIS consists of three components. The first component was an existing program renamed NALCOMIS/I. The second component, NALCOMIS/II, was developed under contract using the waterfall methodology and programming in COBOL. When NALCOMIS/III began forecasting delays and cost overruns using the same approach, program management looked for an innovative development approach to meet budget and schedule. In April 1991, the program manager for NALCOMIS /III adopted a prototyping methodology using an application generator.

NAVAMASSO Prototyping Methodology

The prototyping methodology at NAVMASSO consists of the following steps:

- ***Form a Users' Team:*** A Fleet Design Team (FDT) consisting of experienced aviation maintenance personnel was appointed to work closely with the software developers.
- ***Establish In-House Development Teams:*** Contractors were replaced by five small in-house development teams. These teams have been working closely with experienced aviation maintenance users.
- ***Decomposition of To-Be System into Functional Subsystems and Development Increments.*** NALCOMIS/III is decomposed in ten functional subsystems:
 - Database Administration
 - Flight
 - Maintenance
 - Logs and Records
 - Personnel
 - Asset
 - Data Analysis
 - Technical Publication
 - Reports
 - System AdministrationFull functionality of the subsystems will evolve over five increments.
- ***Gather Iterative Requirements:*** The FDT provides simulated paper screens and interface requirements focusing on user friendliness and extensive on-line help.
- ***Perform Quick Design:*** Developers create screens and interfaces based on the FDT input using a screen generator. The approved screens are then logically put in sequence to provide users with a complete walk-through of the intended system functionalities.
- ***Build Prototype:*** Developers use a database-driven application generator to build the prototype.
- ***Evaluate and Refine Requirements:*** The FDT evaluates the prototype and suggests corrections and enhancements.
- ***Engineer Product:*** When the FDT is satisfied with the functionality provided by the prototype, developers deliver the software to type commanders for testing. When type commanders are satisfied with the product they release it to the

squadrons for use. The system evolves through an adaptive process that eventually converges on a production version that closely meets the users' needs.

- *Users Feedback:* Squadrons make recommendations for software improvements through the type commander.
- *Maintain Product:* Using the same process used in the initial development, the system continues to be adapted to changing user needs.

NALCOMIS/III — Meeting Schedule Constraints and Exceeding User Expectations

NALCOMIS/III increment 1 was developed in five months. It consisted of 157,000 lines of 4GL code for the prototype; the 4GL produced 2.3 million lines of C code for the delivered product. Increment 1 included functionality for seven of the ten subsystems. Performance of the initial release met or exceeded user requirements in 68 out of 71 instances. Overall, the product was much more acceptable to the user than any NALCOMIS product they had seen before.

Increment 2 was completed five months later, *enhancing or completing* the functionality for the same seven subsystems. Increment 3 will be implemented on hardware from a production contract. Approximately 4.8 million dollars were spent on the NALCOMIS/III prototyping effort. This accounts for only 28% of the \$17.5M spent on the entire NALCOMIS/III project. A team of 36 analysts, programmers, and users was able to do what an organization of 85 to 100 contracted programmers with seven layers of management was unable to do.

Increment 2 was subjected to Operational Test & Evaluation (OT&E) from March to May 1992. Although the software functionality suffered only minor discrepancies, increment 2 was judged not operationally suitable for full deployment. The evaluators believe increment 3 is likely for full deployment on operational hardware.

NALCOMIS/III will be subjected to additional operational testing before proceeding to a milestone III decision from the Department of Defense Major Automated Information System Review Council (MAISRC).

Lessons Learned

The following lessons can be learned from the quick delivery of NALCOMIS/III increments 1 and 2:

- Dedication and close cooperation among managers, developers and users is crucial.
- Prototyping enables systems to exceed "pre-defined" functional requirements.
- Prototyping is faster and less expensive than classic software engineering practices.
- Existing Operational Test and Evaluation methodology should be expanded to accomodate evolutionary system development.
- Design documentation should be updated to reflect evolving system design.
- Management must provide a proper environment for prototyping.
- Application generators must be carefully selected.
- Software development contract characteristics should be reevaluated.
- Current DoD hardware acquisition should be expanded to include rapid prototyping and incremental development.

Although the NALCOMIS/III development project can be considered a successful prototyping application, there were a few aspects of the program that could have been smoother. Some of the difficulties could have been avoided by more preparation prior to beginning the prototyping process. Some complications experienced found their roots in difficulty in adapting the Navy development approval process to the rapid prototyping methodology. DoD should implement policies and strategies that promote prototyping.

II. Prototyping as a Development Alternative to Classical Software Lifecycle

In today's climate of budget cuts, military programs must be able to do more with less if they are to survive. For software development projects to be able to accomplish this new standard of efficiency, a more cost-effective software development methodology is required. This new methodology must take advantage of the best productivity tools that current technology has to offer, and apply them in a manner that better adapts to a rapidly changing and financially constrained environment. In many situations, prototyping with application generators, of which fourth-generation languages (4GLs) are a part, offers an opportunity to correct some of the major difficulties caused by the use of traditional software development approach. This section briefly introduces the prototyping concept by contrasting it with the widely used classical development methodology. The readers familiar with these two approaches may skip this section.¹

A. Classical Development Methodology

The classical life cycle methodology is the oldest and has been most widely used. Also referred to as the "waterfall model", the classical methodology consists of six phases. As shown in Figure 1, the methodology is systematic and sequential with the output of one phase serving as the input to the next.²

¹There are a number of software development methodologies reported in literature. Researchers agree that there is no single best methodology to solve the software crisis. This report focuses on the prototyping methodology.

²Pressman, 1987.

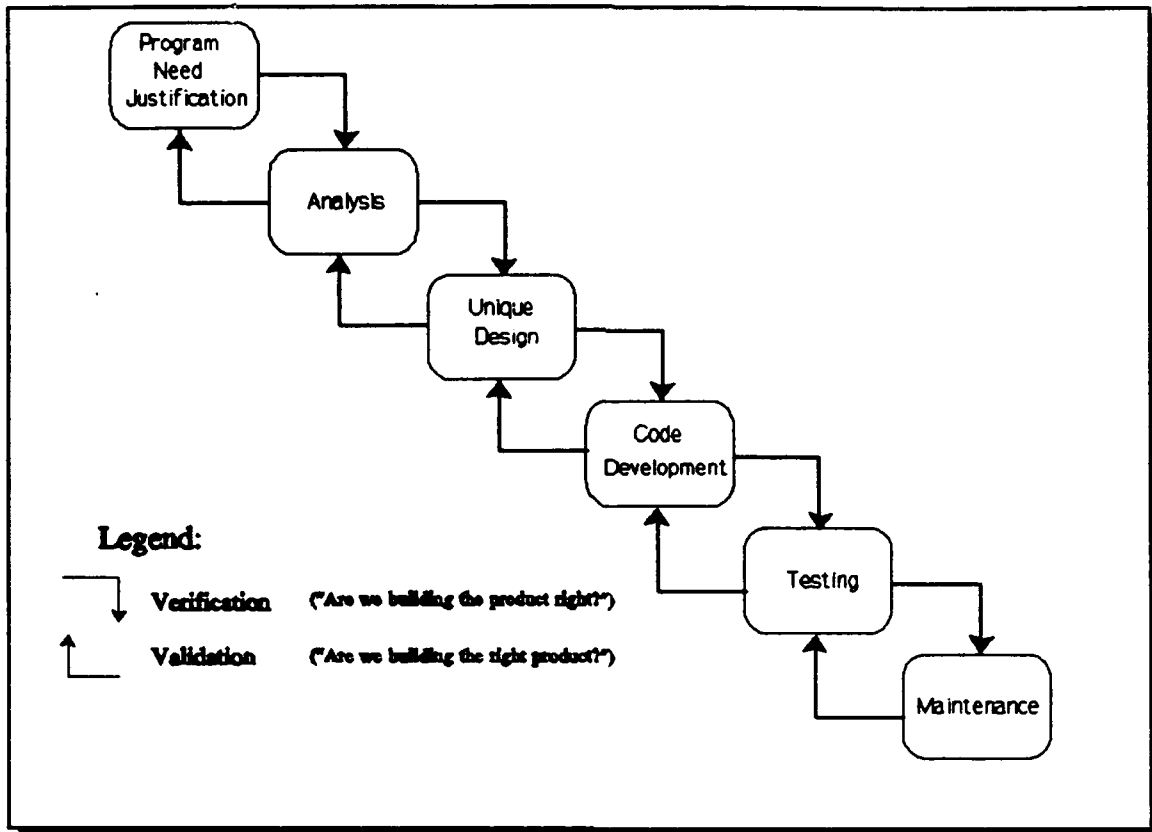


Figure 1. The Classical Waterfall Development Methodology adopted from Pressman, 1987.

- **Program Need Justification:** Organizations explore the problem to be solved and determine the most cost-effective resolution to the problem.
- **Analysis:** Based on the decision to continue and the alternative selected, analysis of the software requirements begins. This phase attempts to document the information domain, required functions, performance, and interfacing needs of the software. Hardware requirements are also determined based on the software specifications.
- **Unique Design:** Often the approach to solving the problem is unique and must therefore be designed from scratch. The design effort focuses on defining and documenting data structure, software architecture, and procedural details.
- **Code Development:** The documents generated in the design phase serve as references for the programmers during the code development phase. The

software resulting from the coding phase is only as good as the design documents it is based on.

- **Testing:** Once completed, the code is tested for correct logic and functionality. Tests are first conducted on functional components and later those units are integrated and tested again. Finally the software is turned over to users for acceptance testing.
- **Maintenance:** Assuming the code passes the testing phase, the software is implemented and becomes operational. As the software is operated, the users will discover bugs or desire enhancements/modifications to the functionality. The maintenance phase of the life cycle is the process of adapting the software to satisfy the new requirements or repair the problems.

While the waterfall model has proven to be appropriate for certain well-defined, highly-structured, large-scale projects in the past, it is not suitable to every development project. The underlying assumptions to the waterfall model are that user objectives are known and fixed and the output from the previous phase in the development cycle is complete and accurate. The consequences of making those assumptions, which are certainly unrealistic in the majority of cases, are programs that exceed budget and time constraints, and do not satisfy user requirements. The classical development approach is very time consuming and often documentation intensive. Mistakes made during the development process using the waterfall model are costly to correct. Infact, it is not unusual to live with a non-critical design flaw rather than spend the money to correct it.

Invariably, the development approach does not work for several reasons. First, and most obviously, requirements evolve over time in response to changes in the environment and technology. Second, even with the best efforts, it is not possible for users to define stable "requirements"; their perceived needs inevitably changes during the course of building and using a system. Finally, if the requirements are well documented and the business process is proven to be effective, often the lack of an appropriate organizational structure to support software development becomes the issue.

B. Prototyping

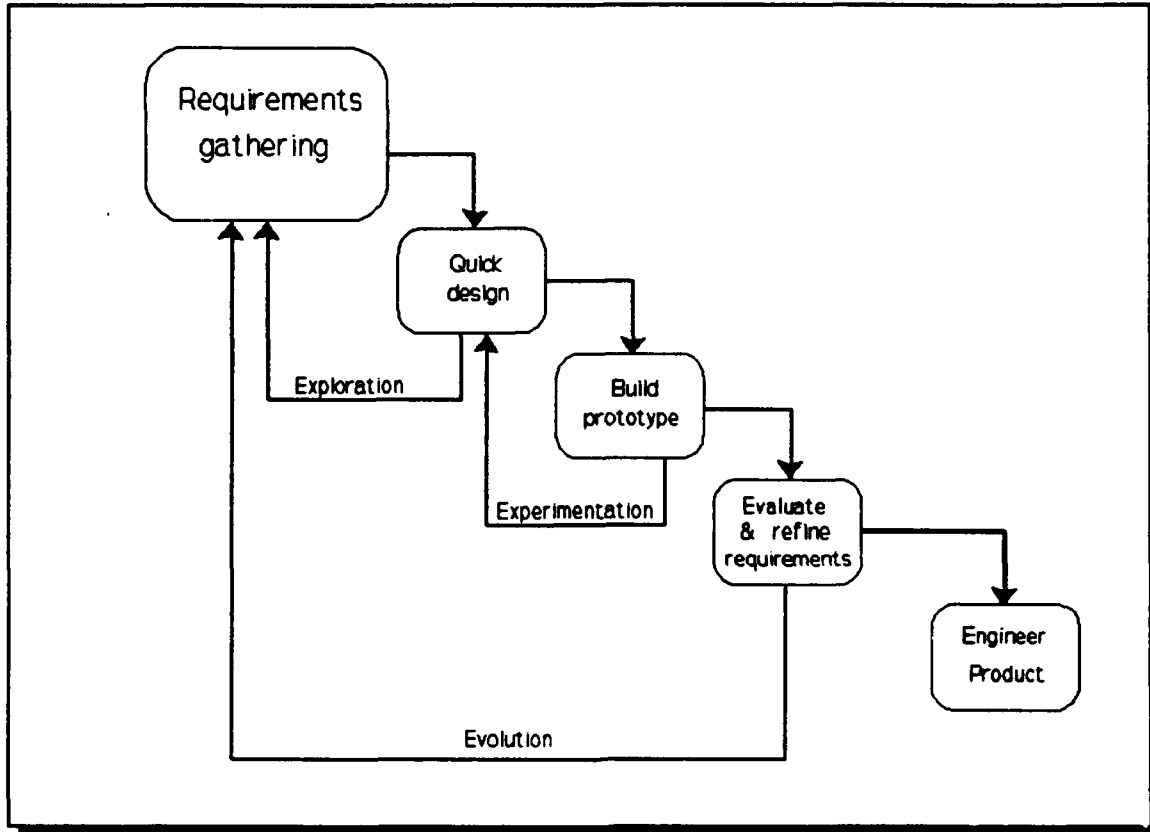


Figure 2. A Prototyping Development Methodology.

Prototyping is a method of developing a working model of the software or software components of the system to be developed. The prototyping approach to software development is perceived to overcome many of the shortcomings of the traditional waterfall model discussed earlier. It lends itself to situations in which system requirements are unclear, volatile, or cannot be communicated easily. As illustrated in Figure 2, prototyping is an evolutionary approach to systems development that consists principally of the following phases:

- ***Gather Functional Requirements:*** The prototyping cycle begins with gathering of users' known requirements and identifying areas needing further clarification.

- ***Design:*** The system developers use the preliminary set of requirements to perform a "quick design" of the prototype.
- ***Build Prototype:*** The prototype is built and turned over to the customer for evaluation.
- ***Evaluate Prototype and Refine Requirements:*** There are no expectations to build the right system the first time; rather, it is used to refine the initial requirements of the users. The users evaluate the prototype for the necessary corrections and enhancements. The cycle continues until users are satisfied with the functionality.
- ***Engineer Product:*** As the requirements are determined they are recorded and the product is built (either through refinements in the prototype system, or translation of the design into a production system written in a different programming language).

The phases of the prototyping cycle shown in Figure 2 can be characterized as exploratory, experimental, or evolutionary. The feedback from the quick design phase back to the requirements gathering phase can be described as exploratory since the purpose of this process is to extract user requirements where few formally exist. The interaction between building the prototype and performing quick design is experimental. The purpose of the prototype building phase is to explore alternative approaches to building the software, testing novel design solutions under different environmental conditions. The interaction between the evaluation process and the requirements gathering effort reflects the evolutionary process that is the essence of the prototyping methodology. This iteration makes it possible for the users' requirements to be incorporated in the system specifications.

The following factors can contribute significantly to the successful application of the prototyping methodology:³

- Users knowledgeable in both the business and the prototyping process.
- Prototype builders knowledgeable of prototyping approaches, supporting tools, and the organization's data resources.
- Predetermination of data element definitions and user interface criteria.

³Wojtkowski, 1990.

Systems developed using the prototyping approach are expected to:

- Provide a clearer definition of project boundaries and scope.
- Experience lower risk.
- Be developed more quickly and less costly.
- Require less user training.
- Promote smoother implementation.
- Be less costly to maintain.

C. Fourth Generation Languages

The iterative characteristic of prototyping requires the ability to quickly build and modify application programs to be cost-effective. This ability generally cannot be supported by third-generation languages such as Ada, COBOL or PL1, due to the excessive cost and time required to make changes in 3GL programs. Fourth-generation languages make it feasible to perform rapid prototyping.

There is no consensus as to what constitutes a fourth-generation language. Products offered in the market often come under the general label of 4GL, but terms such as application generator and integrated CASE (or I-CASE) tool are also used. However, 4GLs are generally non-procedural languages, in that they allow a programmer to specify what needs to be done rather than how to do it. Different 4GLs aim at different intended users with different levels of technical sophistication, ranging from inexperienced end-users to professional data processors developing highly complex systems. Application generators employ 4GLs to facilitate building screens, reports, and data stores.

Ideally, a 4GL should possess the following characteristics:⁴

- A language capable of defining the complete specification of a system, which can then be translated automatically into a program for execution on a selected target computer.
- A set of built-in language functions for defining the type of computational tasks that occur frequently in MIS applications, such as creating screen formats for interactive terminals, defining automatic error checks for input data, generating reports or responses to user queries, and designing "user-friendly" interfaces (e.g., a menu structure).
- Language functions that permit terse specification of a computational task, often best achieved through a nonprocedural language that allows a programmer to specify what task is to be accomplished rather than defining a how-to-do-it procedure. Automatic consistency and completeness checking of a design specification.
- Integrated database management tools for managing the system's database.
- An active central repository, with interactive retrieval capabilities that facilitate access to selected information about the entire system.
- Integrated communication functions for controlling a telecommunications network, handling remote terminals, transmitting data to and from other computers, performing error checks on transmitted data, etc.
- Facilities for managing a secure on-line environment, such as those for keeping track of transactions in their various stages of processing, maintaining a journal of all events within the system, and recovering from a system failure.
- Facilities for integrating the new system with its environment (e.g., other existing applications or network) and keeping track of multiple versions of an application.
- Integrated project management tools for scheduling and coordinating development tasks as defined in the repository.
- A set of design tools with a strong graphical orientation to aid the developer in visualizing relations among system components.
- An assortment of analytical and documentation tools for the support of sound software engineering practices.

⁴Emery et al., 1991.

- Built-in testing facilities (e.g., for generating simulated test data and managing regression testing).
- Capability of generating sufficiently efficient programs to permit the system to handle a high volume of transactions at a feasible cost.

No product currently on the market satisfies all requirements; each of them suffers from at least one of the following limitations:

- Requires a relatively long-term commitment to a single vendor due to the proprietors (non-standard) nature of the product.
- Lacks the functionality to define a complete system within the 4GL's specification language.
- Incapable of integrating the various parts of the existing system.
- Very expensive in terms of hardware requirements and/or software license fees.
- Inefficient in the use of machine resources.
- Immature, without a solid record of successes to lend credibility to the 4GL approach.
- Requires a significantly different approach to software design, and may thus require several months for even an experienced developer to gain full knowledge of their capabilities.

The situation is improving rapidly, however. Some powerful 4GLs are already on the market and proving their worth in developing and maintaining a variety of large MIS applications. Several of them are already valid contenders for use within DoD, and new products or enhancements to existing ones are announced frequently.

The relationship of 4GL and prototyping is illustrated in Figure 3. Typical 4GL applications have shown at least a ten-to-one increase in productivity over those using a lower level language. Their non-procedural nature makes it easier to create and manipulate data. The code is dialogue-like and, therefore, essentially self-documenting. 4GLs are also easier for programmers to learn and use. As a result, programming time is reduced significantly. The rapid development so crucial to prototyping would not be possible without productivity increases offered by 4GLs.

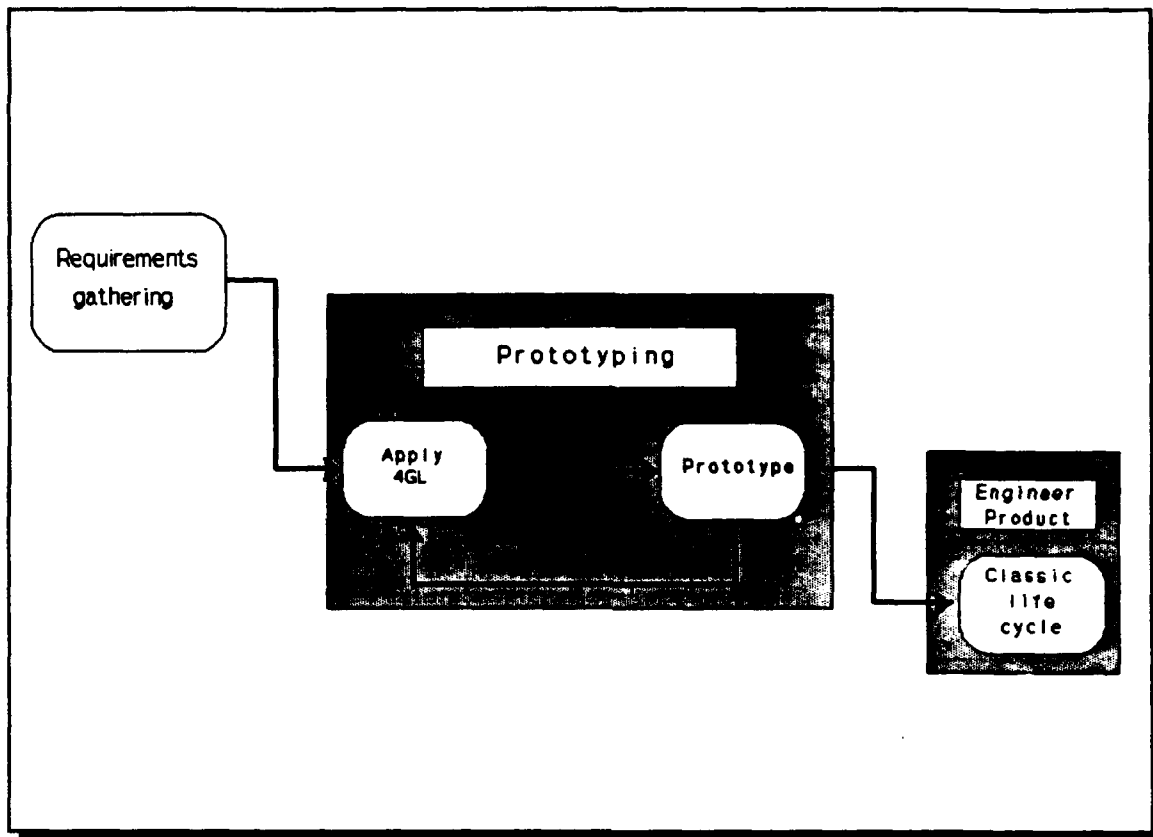


Figure 3. A Hybrid Development Approach.

D. Evaluation Criteria

When evaluating 4GLs for a development project, the following list — although not unique to 4GL — suggests some criteria to be considered.

- **Performance:** refers to the evaluation of the benchmark timing tests. All times should be specified in elapsed wall-clock minutes.
- **Ease of Use:** seeks to appraise the ease with the product can be used on a day-to-day basis. It should take into consideration the skill level of the programmer or user, but it does not include any Data Base Administrator (DBA) functions.
- **Ease of Administration:** addresses the ease with which the product can be administered. The primary considerations are installation, configuration, and performance of typical administrative functions.

- *Documentation:* is critical to software use and maintenance. Quality documentation should be accurate, complete, organized, and easy to use.
- *Customer Support:* evaluates the quality of assistance provided by the software developers. It should reflect the accuracy of their information as well as the timeliness and attitude of the technical support personnel.
- *Data Portability:* encompasses data import and export capabilities.
- *Software Portability:* considers the different platforms that support the product. It should be limited to those computers that have a direct applicability to the processing environment.
- *Effective Use of Resources:* is a measure of the effectiveness with which the product employs the computer's resources. The effectiveness should not be a measure of efficiency. Instead, the effectiveness should be a subjective measure of how well the product takes advantage of the capabilities of Operating System and the features of the hardware. Of primary importance are disk space, system memory, and CPU requirements.

E. Summary

Experience with 4GLs used in support of prototyping suggests that significant productivity gains can be achieved to enable organizations to "do more with less." 4GLs facilitate managing software applications by producing more consistent documentation and reducing the time and effort required to develop, modify and maintain software applications. Employed properly, prototyping supported by 4GLs should result in lower development costs and a higher quality end-product for lower life cycle costs.

III. Transition to Prototyping: The NALCOMIS Case

This case study illustrates the use of prototyping in DoD. It reports the experiences from the Naval Aviation Logistics Command Management Information System (NALCOMIS) effort in developing an information system that automates the maintenance procedures of Naval aviation units. This system, known by the sponsoring command as NALCOMIS/III, is the third component of the entire information system whose first two parts were developed under contracts using the classical development approach. This section offers some factual background useful for understanding lessons learned from the DoD prototyping effort with NALCOMIS/III.⁵

A. Mission

In 1959, the Chief of Naval Operations established the Naval Aviation Maintenance Program (NAMP) to integrate aeronautical equipment maintenance procedures and related support functions. The NAMP distinguished three different organization levels — individual squadron, headquarter level, and depot level — at which aviation maintenance was to be performed based on the increasing complexity of maintenance tasks. By assigning particular tasks to the appropriate levels, the Navy can better achieve optimal use of resources.

The Naval Aviation Maintenance and Material Management (AV-3M) System (an information system) grew out of the NAMP in 1965 as an attempt to modernize data collection and information reporting for aviation activities. Because of the timeframe in which AV-3M was introduced, the Navy had few technological resources to assist with

⁵Background information on the NALCOMIS program was obtained from Allen, 1988.

this task. Therefore, the primary benefit AV-3M had to offer was the standardization of the manual processes.

The NALCOMIS project, established by the Chief of Naval Operations in 1975, was the next attempt at modernizing the aviation maintenance program. There are four principle objectives for the system:

- Increase aircraft material readiness.
- Improve the efficiency of aircraft maintenance and supply support organizations.
- Improve the quality and timeliness of aviation data reported upline, and
- Reduce overhead labor and paperwork costs required to operate and execute the NAMP at the local level.

B. Key Organizational Players

As depicted in Figure 4, there are many organizations involved in the NALCOMIS project. Their roles and functions are briefly described below.

1. Naval Air Systems Command (NAVAIR) PMA-270

Located in Crystal City, Virginia, PMA-270 is responsible for management of the overall program. This NAVAIR office enforces budget and schedule constraints while ensuring that sufficient resources to adequately accomplish the development. The Program Manager also has the responsibility to ensure continued congressional support by successfully satisfying all Major Automated Information System Review Council (MAISRC) requirements.

2. Navy Management Systems Support Office (NAVMASSO)

Located in Chesapeake, Virginia, NAVMASSO became the central design agency for NALCOMIS in May 1984. This office initially acted as the Navy liaison between the contractors and users. NAVMASSO has replaced the contractors as the developers of the NALCOMIS/III.

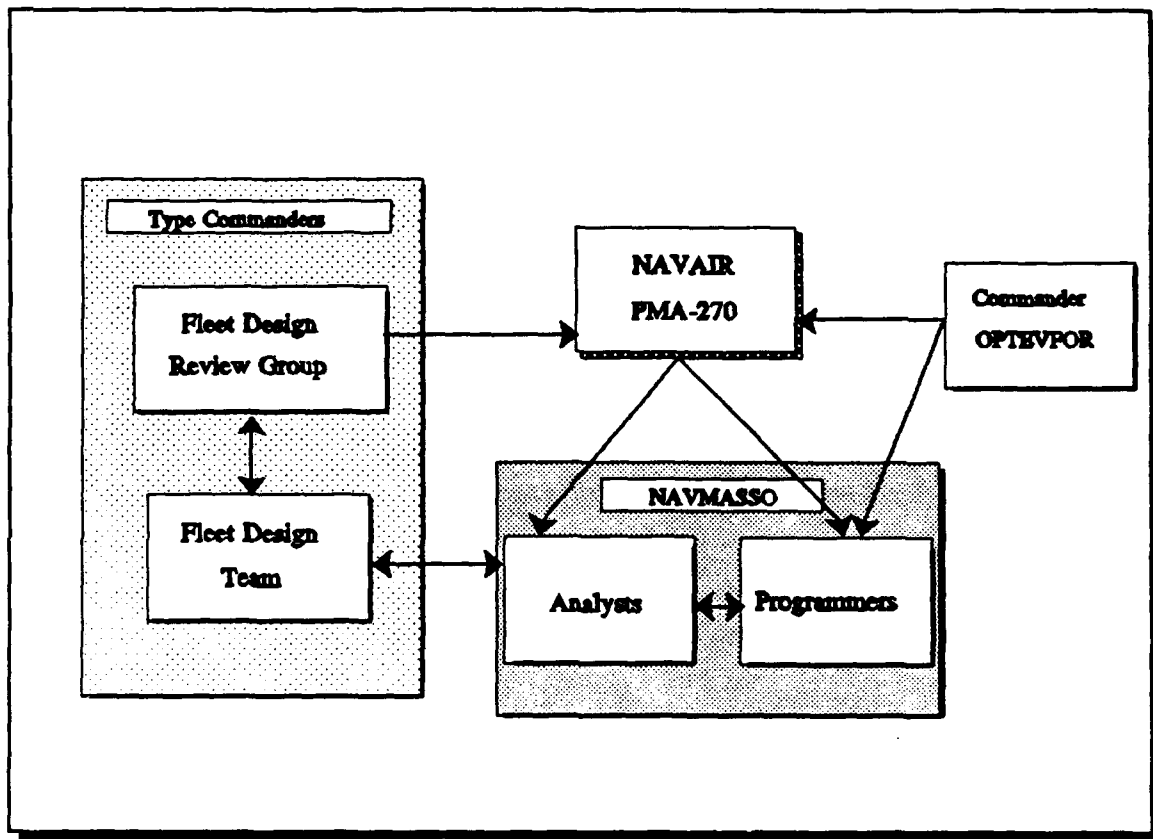


Figure 4. Key Organizations Involved in NALCOMIS/III Development (arrows represent communication channels)

3. Type Commanders

The Type Commanders are the high level organizations that represent the users. There are five type commanders representing Atlantic, Pacific, and Reserve aviation, Training and Naval Air Systems Command units. They are located in Norfolk, Virginia; San Diego, California; New Orleans, Louisiana, Corpus Christi, Texas; and Washington D.C. respectively. The Atlantic Type Commander invited the Second Marine Air Wing (2nd MAW) located in Cherry Point, North Carolina to participate in the NALCOMIS/III development providing Marine Corps representation.

4. Fleet Design Team (FDT)

The FDT, comprising of senior enlisted aviation maintenance sailors and marines from each of the type commands, is the user group that provides requirements and design feedback to the developers. When the team is activated, these members are assigned temporary additional duty at NAVMASSO.

5. Fleet Design Review Group (FDRG)

The FDRG, consisting of aviation maintenance officers at each of the type commands, reviews all major decisions made by the FDT and provides additional guidance.

6. Commander Operational Test & Evaluation Force (COMOPTEVFOR)

COMOPTEVFOR performed the Operational Test and Evaluation for NALCOMIS/III. The Program Manager chose COMOPTEVFOR to conduct the testing because the organization was perceived to be the most proficient in the Navy at testing software systems. COMOPTEVFOR specializes in testing weapons systems.

C. Background

NALCOMIS was to be developed in three main components with each concentrating on a single organization level identified by NAMP. Automating one level at a time would provide fleet users with an interim system until a fully NAMP supportable system could be developed. In this report the different components will be referred to as NALCOMIS/I, NALCOMIS/II, and NALCOMIS/III.

1. NALCOMIS/I

NALCOMIS/I is a new title for an existing application previously known as the Status Inventory Data Management System (SIDMS). SIDMS application was developed on Harris H-300 hardware in 1981 under the design guidance of Commander Naval Air

Atlantic. The application was adapted to run on Shipboard Non-Tactical ADP Program (SNAP) hardware in 1984 and renamed the NALCOMIS Repairables Maintenance Module (NRMM). NALCOMIS/I is being used to support the Aircraft Intermediate Maintenance Departments (AIMDs) and Supply Support Centers (SSCs) until NALCOMIS is fully developed.

2. NALCOMIS/II

As with NALCOMIS/I, NALCOMIS/II is directed toward the intermediate level activities and is intended to include the aviation maintenance functionality that was left out of the supply-oriented NALCOMIS/I by providing automated data collection and on-line data processing capabilities to the AIMDs and SSCs. The development chronology of NALCOMIS/II is shown in Table 1. NALCOMIS/II, consisting COBOL programs on Honeywell DPS-6s, was operationally certified in March 1989 — more than three years after software testing began.

September 1985	Best-effort contract awarded to Eldon Associates ⁶ for NALCOMIS/II & III development.
February 1986	NALCOMIS/II software testing began.
June 1986	User acceptance testing of NALCOMIS/II began at Marine Aircraft Group 14 (MAG-14).
March 1989	NALCOMIS/II software was operationally certified.

Table 1. NALCOMIS/II Development Chronology

⁶Fictitious name.

The development and implementation of NALCOMIS/II provided two important lessons that have been applied to NALCOMIS/III. First, the importance of involving users early in the development process became obvious when the contractor introduced the software to the users and encountered high user frustration. Second, although it was not realized until several months later, the inappropriateness of the waterfall development methodology caused severe budget and schedule overruns. The difficulties encountered with the adoption of the waterfall model will be discussed later.

D. Functional Requirements of NALCOMIS/III

Just as NALCOMIS/II has automated the AIMDs, NALCOMIS/III is intended to eliminate numerous man-hours spent on the manual collection, processing, and reporting processes supporting aviation maintenance at the squadron-level.

NALCOMIS/III is expected to interface with NALCOMIS/II, enabling the automated exchange of information among the squadrons, AIMDs, and SSCs. The initial analysis of NALCOMIS/III identified ten subsystems, as depicted in Figure 5. A brief description of each of the subsystems follows:⁷

- *Database Administration:* provides system-level support tables of squadron baseline, system security, and maintenance data.
- *Flight:* collects and processes flight-related data and provides these data to other subsystems.
- *Maintenance:* collects and processes maintenance-related data and provides these data to other subsystems.
- *Logs and Records:* establishes and maintains configuration profiles on aircraft, propellers, engines, modules, and components assigned to the squadron.
- *Personnel:* provides the ability to track specific information on selected personnel assigned to the squadron.
- *Asset:* tracks information on survival, safety, and other aviators' gear allocated

⁷NAVMASSO Document J-004 FD-002B, 1992.

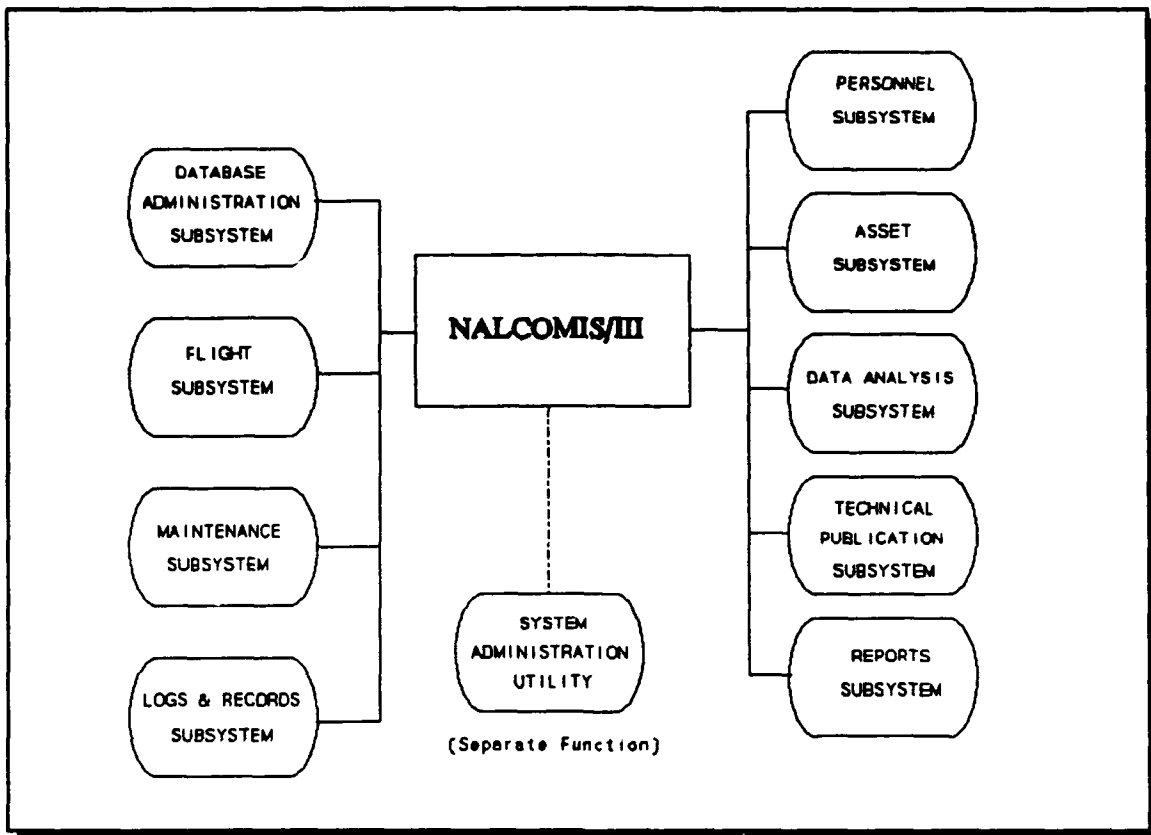


Figure 5. NALCOMIS/III Subsystems.

to the squadron.

- **Data Analysis:** provides analysts the ability to review and correct each flight and maintenance record prior to extracting these records for a supporting system.
- **Technical Publications:** provides the ability to manage the squadron's assigned aeronautical technical publications.
- **Reports:** generates predefined standard reports.
- **System Administration:** provides the system administrator the ability to maintain the squadron's NALCOMIS system.

The developers did not build all ten functional subsystems at once. The FDT prioritized subsystems, identifying those that would be required for an initial operational

system. Developers analyzed those requirements to determine if other functions were necessary to meet the FDT specifications. Software packages were identified by increment number to indicate the level of functionality to be included. There will be several releases of the same increment in order to correct deficiencies, enhance functionality and incorporate lessons learned from the prototype into the current release. The fully functional NALCOMIS/III software will be developed over five increments.

E. Initial Development Strategy of NALCOMIS/III: NALCOMIS/II Difficulties Revisited

As with many DoD management information systems, the waterfall development methodology was used to build NALCOMIS/II and begin NALCOMIS/III.

1. Difficulties Encountered with the Classical Development Methodology

Figure 6 illustrates the NALCOMIS/II development strategy. The need for an automated system was justified by the need to eliminate time consuming manual tasks required for aviation maintenance. Eldon Associates offered the lowest bid to develop the system and won the contract for a duration of five years. The users provided their requirements to NAVMASSO, which in turn determined technical feasibility and interpreted the requirements to the contractor. The later completed analysis of the problem by compiling a document of user requirements and started to design the system. The design was communicated to the users two foot in a stack of documents called the Functional Design Requirement Document. Upon the approval of the Type Commanders, the contractor began coding and testing. One NAVMASSO employee admitted that the NALCOMIS/II programs that were delivered did compile cleanly. However, proper testing did not begin until the software was delivered to the Marine Air Group (MAG-14). NALCOMIS/II software is currently in the maintenance phase.

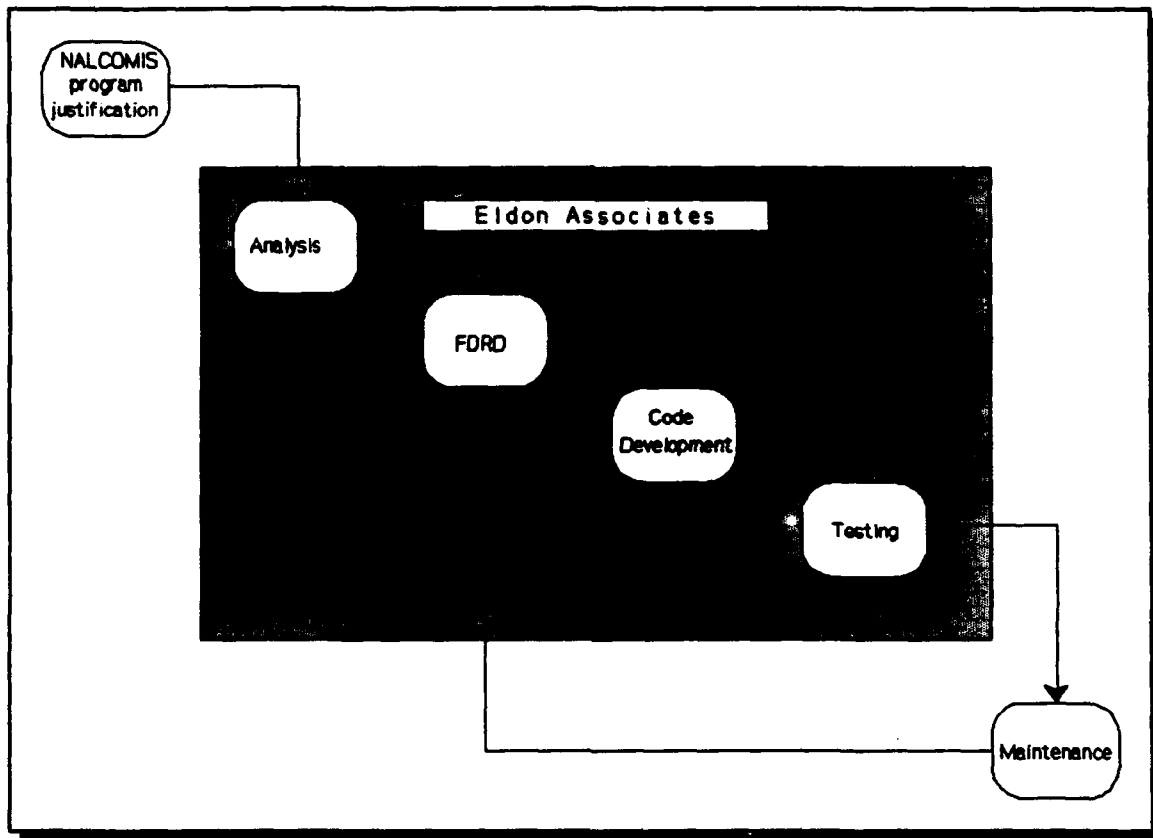


Figure 6. NALCOMIS/II development.

The waterfall SDLC did not work. For several reasons enumerated below, users experienced frustration with the new system:

- Users provided minimal input during the requirement analysis phase. Once requirements were documented, users were unable to provide feedback on the overwhelming quantity of documentation.
- Design reviews, when they occurred, were held with upper management rather than with the future system users.
- Coding was conducted off site by a third-party contractor using a third-generation language (COBOL).
- Adequate testing of the software was not conducted prior to implementation. Numerous errors were discovered by the users during implementation. User acceptance testing began after the software was installed at MAG-14.

- Maintenance of the software (just to operationally certify NALCOMIS/II) was so costly it took three years and used the finances budgeted for both NALCOMIS/II maintenance and NALCOMIS/III development.
- The lack of user involvement throughout the NALCOMIS/II development process proved too costly in terms of dollars and time.

2. First Development Experience with NALCOMIS/III

Since NALCOMIS/I and NALCOMIS/II were operational, remaining funds were used to maintain them. In July 1987, the NALCOMIS program began experiencing financial difficulty. As a result, the development of NALCOMIS/III was suspended indefinitely. Because of cost overruns incurred by NALCOMIS/II, the development effort for NALCOMIS/III did not resume until November 1990. At this time, as the five-year contract with Eldon Associates expired, ActionWare⁸ —again, as lowest bidder — won the NALCOMIS contract. The new contractor followed the development approach initiated by its predecessor. Since the documents created by Eldon Associates containing user requirements and system specifications for NALCOMIS/III were already in place, ActionWare continued with the coding phase, using COBOL on the Honeywell DPS-6 (SNAP I).

The program management had realized the importance of user involvement in the system development process from the difficulties of NALCOMIS/II development. The Program Manager sought user representation by asking each of the Type Commands to send a representative to provide inputs to the contractor via NAVMASSO. Five senior enlisted (E7-E9) personnel experienced with Naval aviation maintenance formed the Fleet Design Team in November 1990.

Although the users now had an avenue to express their concerns, their comments were not always incorporated in the development because they were filtered by NAVMASSO before they reached the contractor.

⁸Fictitious name.

ActionWare used primarily COBOL for the application programs and INFORMIX/DBMS for the database portions of the system. ActionWare estimated that NALCOMIS/III Increment 1 would require one million lines of code to complete. The contractor had access to the COBOL code that its predecessor had created during the initial development stages. ActionWare intended to use C or the INFORMIX/DBMS wherever COBOL could not be conveniently used. NAVMASSO employees began to fear a hodge-podge of code that would be a nightmare to maintain.

In January 1991, management became aware that costs were growing, the schedule was slipping, functionality began to shrink, too much time was spent negotiating the terms of the contract, and the government/contractor/government turn-around was too slow.⁹ Concerned by these events, and anxious to keep the implementation schedule, the Program Manager was forced to devise a more cost-effective plan of action. As a first action, the DPS-6 minicomputer was replaced by the Bull DPX/2 micro-computers. This was a practical move to reduce hardware costs and eliminate the need for computer rooms at all operational units. This move also resulted in establishing the UNIX Operating System environment rather than the very proprietary General Comprehensive Operating System (GCOS) that the DPS-6 used.

3. Transition to Prototyping with an Application Generator

The more NAVMASSO employees learned about the potential benefits of 4GL, the more convinced they became that the task had a greater chance of being accomplished with INFORMIX/4GL than with COBOL. ActionWare, however, showed no signs of wanting to make the transition to the 4GL. This hesitancy may be attributed to the resistance to disregard the sunk cost of the COBOL code already produced. ActionWare claimed the application was 80 percent complete. NAVMASSO believes it was less than 50 percent complete of the stipulated requirements.

⁹Obtained from an interview with NAVMASSO employees on January 31, 1992.

In January 1991, the head of the NAVMASSO Aviation Systems Directorate approached his Commanding Officer proposing to him to discontinue the NALCOMIS/III development contract, and continue the effort in-house using INFORMIX/4GL.

A NALCOMIS Program Review was held at NAVMASSO on January 23, 1991. Realizing the program was not progressing as it should, the Program Manager asked NAVMASSO to investigate alternatives to deliver the NALCOMIS/III software on schedule in August 1991. NAVMASSO identified the following alternatives, and their respective potential repercussions:

- *Alternative 1:* Status Quo (i.e., proceed with the current contractor), formally estimated to cost \$1.4M. NAVMASSO believes, however, that this avenue would eventually lead to complete failure.
- *Alternative 2:* Add funding (i.e., proceed with the current contractor with additional funding to cover cost overrun); the total cost was expected to amount to \$1.8M. NAVMASSO believes, however, that additional funding would not help in resolving current difficulties.
- *Alternative 3:* Move NALCOMIS/II software maintenance from ActionWare into NAVMASSO; leave NALCOMIS/III with ActionWare. This alternative would cost \$1.4M and cause a four to six month delay.
- *Alternative 4:* Move NALCOMIS/III from ActionWare into NAVMASSO; place all NALCOMIS/II coding with ActionWare. This alternative could be accomplished within the current budget.
- *Alternative 5:* Competitive development effort using NAVMASSO and ActionWare. ActionWare proceeds as in Alternative 1; in parallel, NAVMASSO proceeds as in Alternative 4. Progress of both efforts would be reassessed in April 1991; best approach is continued, the other is canceled. According to NAVMASSO. The cost of this alternative would be approximately \$1.8M.

Choosing a 4GL to develop applications software would mean embracing a methodology that deviates from common typical DoD practices. After some consideration of the alternatives, the Program Manager got the approval from his command for Alternative 4.

Since one of the primary justifications behind the replacement of the classical development approach was time savings, it logically followed that rapid prototyping would be the necessary development methodology. There was no time for developers or users to glean requirements out of outdated documentation. The prototyping methodology was chosen for NALCOMIS/III because it was perceived to offer the greatest opportunity for the system to evolve within the given time constraints. overrun.

.

.

.

.

IV. NALCOMIS/III: Prototyping with an Application Generator

A. Hardware Environment

1. Prototype Hardware

The NALCOMIS/III prototype was developed on the Bull DPX/2 Model 220 mini-computer with a UNIX operating system. The Central Processing Unit is a 32 bit Motorola 68030 microprocessor with an operating speed of 25 Mhz. The DPX/2 has 16 MB of memory with two 675 MB Internal disks and a 150 MB internal streamer/tape drive.

BI-LINK Portable microcomputers act as terminals. These 386 processors can operate either as a NALCOMIS/III terminal or as a stand-alone personal computer. BDS-7 dumb terminals and Zenith Supersport 286e Laptop computers are alternatives for use as terminals. Some BDS-7 terminals have been implemented in squadron workcenters due limited hardware resources; however, no lap-top computers have been used as terminals.

Two types of printers are used with the prototyped system:

- Impact Line Printer Model 970 used to print formal maintenance documents.
- Screen Printer Model 4/22 used to print screen dumps and informal working copies of maintenance documents.

2. Operational Hardware

The operational hardware has not yet been determined. The initial Request For Proposals (RFP) was issued on February 15, 1992. As of late August 1992, the contract had not been awarded.

The RFP called for:¹⁰

"a base configuration consisting of a computer functioning as a host utilizing a POSIX compliant UNIX operating system. User workstations will be connected via an Ethernet IEEE 802.3 10base5 Local Area Network (LAN), modem and direct connection to RS-232-C ports."

The RFP also requires a live test of the existing application software and database on the proposed hardware.

B. The Development Process

The application generator, INFORMIX/4GL was already in place as it accompanied the INFORMIX/DBMS purchased during the initial stages of NALCOMIS/III development. The analysis and design for NALCOMIS/III had been produced in the FDRD by Eldon Associates in 1986. However, requirements had changed in the four-plus years since. Additionally, there was little time for NAVMASSO to digest several thousand pages of documentation. NAVMASSO adapted a development methodology they believed would allow a quicker, more accurate extraction of system requirements. The prototyping process used for NALCOMIS/III development is illustrated in Figure 7.

- *Gather Iterative Requirements:* The FDT provided paper screens and interface requirements focusing on user friendliness and extensive on-line help.
- *Quick Design:* Developers created screens and interfaces based on the FDT input

¹⁰Commerce Business Daily Weekly Release in January 1992.

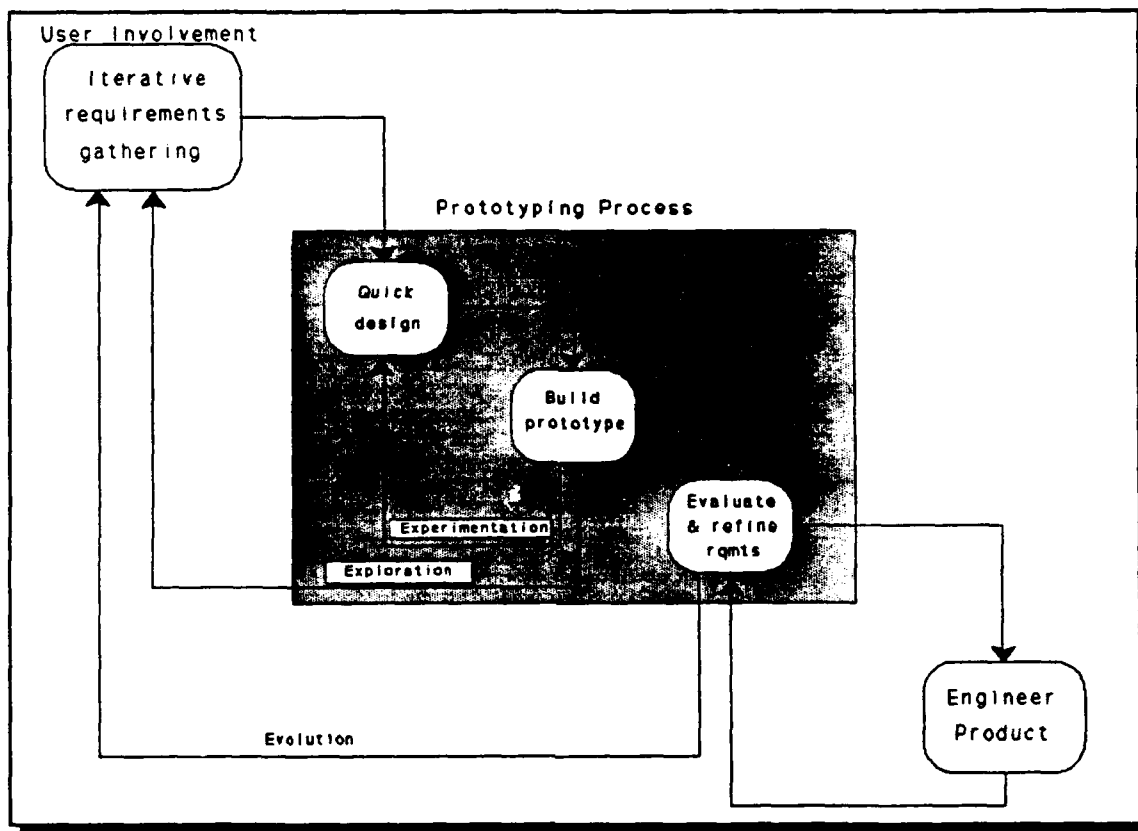


Figure 7. NALCOMIS/III Prototype Development Methodology.

using INFORMIX/4GL.¹¹ The screens and interfaces eventually formed functions.

- *Build Prototype:* When sufficient functionality had been designed, a prototype was built.
- *Evaluate and Refine Requirements:* The FDT evaluated the prototype and suggested corrections and enhancements and the cycle continued.
- *Engineer Product:* When the FDT was satisfied with the functionality the component became part of NALCOMIS/III.

No matter how competent the FDT is, such a small group cannot cover all aspects

¹¹This process took minutes with INFORMIX/4GL compared to hours with COBOL.

of Naval aircraft maintenance. A larger, more extensive group would be more difficult to manage when providing requirements to the developers. An extended group of users known as "Alpha sites" (shore-based squadrons) provided additional insight to the software developers after the FDT. Nineteen squadrons from all over the country and representing all different types of aircraft and operations were identified to be the first sites to implement increment 1. Five different operational sea-going (carrier-based) squadrons were designated "Beta sites" to implement increment 2 along with the Alpha sites.¹²

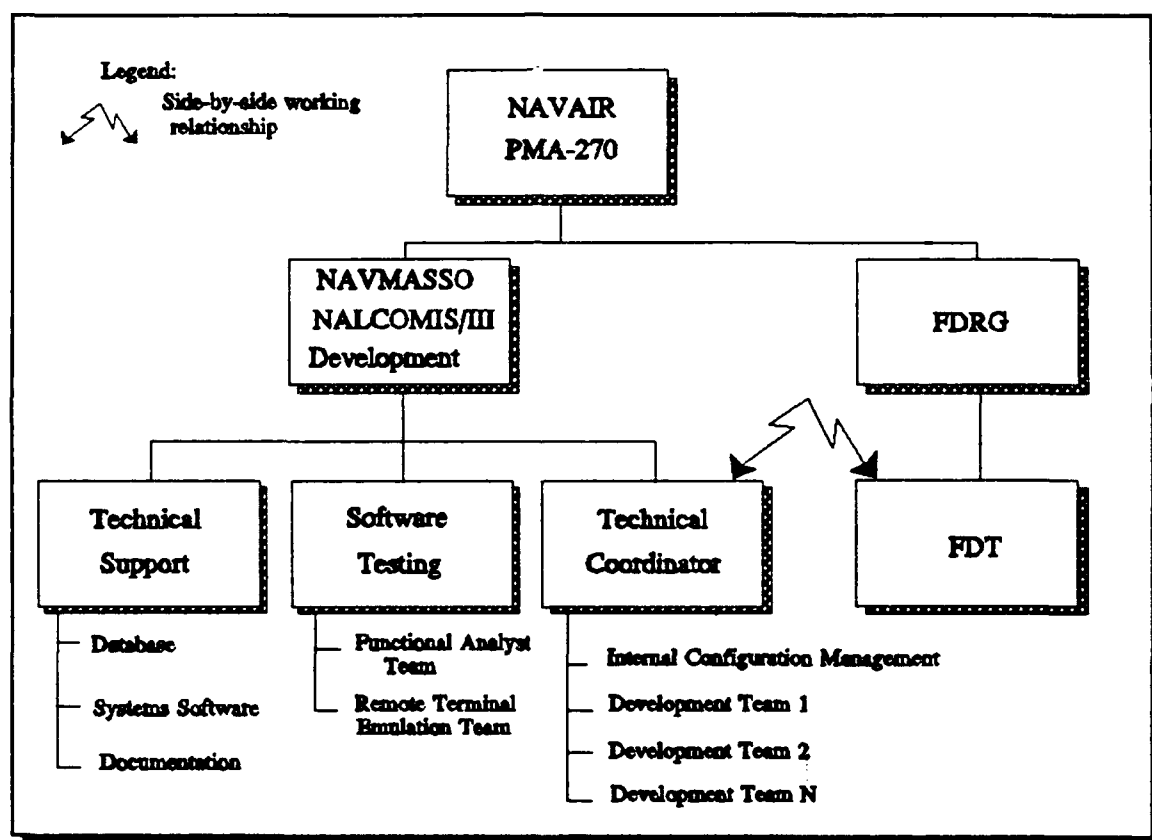


Figure 8. NALCOMIS/III Development Organization

¹²Ten sites were identified as Beta sites; however, the number had to be reduced due to limited hardware resources.

The interaction of the organizations involved with the development is depicted in Figure 8. Interaction between the FDT and the developers was constant. Major design decisions were evaluated by the FDRG. The users represented by the FDT were not involved at one stage of the development process; rather, they became part of the entire development effort due to the iterative nature of the requirements gathering process. This involvement would not have been possible without the rapid productivity provided by INFORMIX/4GL.

C. An Assessment of the NALCOMIS/III Prototyping Approach — Current Status

1. Software Development Status

As of August 1992, NALCOMIS/III has completed software increment 1, increment 2, and some of increment 3. Increment 1 included capabilities for database administration, flight, maintenance, logs and records, asset data analysis, and reports subsystems.¹³ Increment 2 enhances and completes the functionalities of increment 1. Increment 3.0 has been developed except for hardware dependent modules. Those modules are expected to be completed as soon the operational hardware becomes available.¹⁴ Increment 1 was produced in five months and consisted of 157,000 lines of 4GL code; the 4GL code generated approximately 2.3 million lines of C code. NAVMASSO estimated the 4GL to be commensurate to approximately 1.4 million lines of COBOL code using a nine-to-one equivalency ratio¹⁵. Performance of the initial release met or exceeded FDRG/FDT requirements in 68 out of 71 instances. Overall,

¹³In accordance with the Functional Description for NALCOMIS/III some subsystems achieved their full functionalities, while others were only partially implemented as scheduled at increment 2. See Figure 5.

¹⁴The tardiness of the hardware selection is due to the very time consuming DoD procurement process.

¹⁵Pressman suggests a ratio of ten or twenty-five to one may be more accurate.

the product was much more acceptable by the user than any NALCOMIS product they had seen before.

2. Development Costs

Approximately \$4.8 million was spent on NALCOMIS/III development with the prototyping approach. This only accounts for 28% of the \$17.5M spent on the entire NALCOMIS/III project.¹⁶ Thirty-one NAVMASSO employees¹⁷ and five FDT members were dedicated to the NALCOMIS/III development effort. These individuals were organized into teams as shown in Figure 8. They began building the application in April 1991 after three weeks of training COBOL programmers in INFORMIX/4GL, UNIX, and C programming language.¹⁸

3. Schedule

Started in April 1991, increments 1 and 2 and the majority of increment 3 are completed 17 months later. Although the developers would have liked to have another month, the teams met their first deadline in September 1991. A team of 36 analysts, programmers, and users was able to do what an organization of 85 to 100 contracted programmers with seven layers of management was unable to do. The schedule continues to remain demanding.

The prototyping process led to a number of significant changes in the management process, compared to the conventional waterfall methodology. The NALCOMIS

¹⁶Of the \$17.5M, \$3.6M were spent for Eldon Associates, \$.7M for ActionWare; other costs are attributed to training, implementation, and other administrative costs.

¹⁷Nine of the 31 individuals have been contracted from another government agency and work side-by-side with the NAVMASSO employees. Six of the 31 have been contracted from a local civilian consulting agency.

¹⁸INFORMIX/4GL generates C code. C is a third-generation language.

schedule was tight; too much time had already been spent and perceived as wasted by trying to produce the system using the waterfall methodology. NAVMASSO believed it would be counter-productive to salvage previously documented requirements and design specifications. Since the FDT member often sat next to the programmer providing alternative solutions as they went, requirements could change five or six times within an hour. Furthermore, as requirements evolved, there was no time to incorporate them in the existing documents. The command decided to start over with the new development paradigm. Although user manuals and program and system specifications are updated regularly, design documentation is not. The impact of the lack of design documentation on the NALCOMIS program has yet to be determined.

4. Testing and Evaluation

Initially the program was scheduled for MAISRC Milestone 3 review in April 1992, with a cost-benefit analysis and a favorable Operational Test & Evaluation (OT&E).¹⁹ As the only increment available at that time, increment 1 was subject to OT&E in January 1992. Conducted by COMOPTEVFOR, OT&E ended within a week with a "deficiency" rating because of unsatisfactory operational effectiveness and suitability. Increment 1 refused input, provided erroneous output, and locked-up during busy processing periods. As a result, MAISRC Milestone 3 was tentatively rescheduled for July 1992.

OT&E resumed with increment 2 in March and ended in May 1992. COMOPTEVFOR determined that NALCOMIS/III was "operationally effective but not operationally suitable". Since increment 3.0 will be implemented on new operational hardware, the evaluators determined NALCOMIS/III to be "potentially operationally suitable with increment 3.0". It is important to note that COMOPTEVFOR was evaluating NALCOMIS/III against the Mission Needs Statement for the final hardware requirements and other standard checklists without consideration of the incremental

¹⁹See NAVDAC PUB 24.2 for a description of MAISRC Milestones.

development approach being employed for NALCOMIS/III development.

As a result of the rapid development promoted by the 4GL, increment 2 had many capabilities that were not planned to be introduced until as late as increment 5.

Users responded to the Chief of Naval Operations with fervent support of the software, explaining that "to date, NALCOMIS/III has, in virtually every respect, outperformed (the users') greatest expectations," and "fleet/squadron enthusiasm for the achieved benefits already far outweighs any shortcomings...".²⁰ Another user group regarded NALCOMIS/III increment 2 as "an unsurpassed string of successes".²¹ The OT&E will be updated in late 1993. MAISRC Milestone 3 has been tentatively rescheduled for early 1994.

5. Training Adequacy

a. Programmers' Training

The alternative chosen by the Program Manager required the developers to be knowledgeable in INFORMIX 4GL, UNIX, and C. NAVMASSO had no resident expertise in any of these areas. The prior approach required COBOL programmers and so there was an abundant supply of knowledgeable COBOL programmers. NAVMASSO programmers, though somewhat familiar with INFORMIX/4GL, had to become proficient in that language, as well as the C code generated by the 4GL and the UNIX Operating System, if they were to complete the coding effectively and efficiently.

The Program Manager funded training for all NAVMASSO employees involved in the NALCOMIS development — from managers to programmers — in UNIX, INFORMIX, and C. The developers spent three weeks in formal classroom training.

²⁰Commander Naval Air Atlantic message to Chief of Naval Operations dated 23 July 1992.

²¹Commander Naval Air Pacific message to Chief of Naval Operations dated 29 July 1992.

Additionally, PMA-270 hired an INFORMIX consultant to be involved in the programming effort. Initially the consultant was on site at NAVMASSO full time, providing assistance to the programmers as they hit snags in their coding. The time he spent physically at NAVMASSO grew less and less, and after five months he was only used on an on call basis. Although the consultant's expertise was expensive, NAVMASSO employees assessed this assistance as an invaluable contribution to the projects success.

Even though no one had experience in UNIX or C, the training time for 4GL was found to be much less than required for a typical third-generation language. The ease of training can be partially attributed to the English-like nature of the language. Another reason for training success was the skill level and background of the programmers involved in the project.

b. Users' Training

Implementation of the prototype at the Alpha and Beta sites proved that one to two weeks of "over-the-shoulder" training was adequate for system users. COMOPTEVFOR determined the System Administrator training to be inadequate during the OT&E, as most System Administrators lacked the basic skills to trouble shoot even minor problems. When a new site is implemented, the designated System Administrator, usually an E6-E7 aviation maintenance administration specialist with minimal computer experience, receives two weeks of formal classroom training. A formal System Administration course is currently being updated to provide training in diagnosis, troubleshooting, and repair of hardware and LAN-related problems.

6. Management Effectiveness

The NALCOMIS/III development organization, illustrated in Figure 8, consisted of seven teams of three to four programmers. Initially, no individual team leaders were appointed in keeping with Total Quality Management (TQM) philosophy. In most

instances, the lack of a team leader was detrimental to the effort; eventually leaders emerged, and were later formalized by management.

NAVMASSO employees assigned to the NALCOMIS/III development effort were well-educated, dedicated professionals. Their sense of dedication and high morale were critical to the successful application of the prototyping technique, especially since milestones were scheduled with very little flexibility. Since NAVMASSO was aware of the risk that the NALCOMIS program could be eliminated for not meeting the expected milestones, demanding work hours were necessary. Overtime was abundant; leave was scarce. If these conditions persist, morale could suffer and lead to distressing effects on future NALCOMIS/III development and software maintenance.

7. Benefits Analysis

According to a study conducted by an office independent from NAVMASSO, NALCOMIS/III would provide between \$105K and \$195K in savings per aircraft squadron per year, primarily from reduced manual labor.²² The study also projects a total potential savings of \$249M to \$696M through the year 2008.

Most important, early use of the system suggests that NALCOMIS/III does improve the management decision-making process by providing the following functional enhancements:

- Improved data collection and accuracy.
- Improved technical publication control and updating.
- Improved tracking of technical directive information.
- Improved identification of recurring maintenance problems.
- Improved pilot/aircrew flight data reporting/record keeping.
- Improved standard asset management at the squadron level.
- Improved configuration control of aircraft, installed engines, components, and

²²Office of Technical Assistance, December 1991.

support equipment.

- Improved maintenance through automated tracking of aircraft and support equipment scheduled maintenance.
- Improved upline management information through timely automated data from squadron detachments.

Full mission capability of aircraft is expected to increase from 1.18 to 1.94 percent due to improved supply response and reduced administrative delay time.

V. Lessons Learned

A. Dedication of Managers, Developers and Users is Crucial

The early success of NALCOMIS/III can be attributed to the unprecedented dedication of managers, developers, and users. The Program Manager open-mindedly explored new, unproven software development approaches to salvage a badly flawed program, turning NALCOMIS into a potential Gold Nugget success story. The Commanding Officer and staff of NAVMASSO took an unprecedented risk in attempting the development in-house with unfamiliar technology, proving that the government has far more than adequate resources and skill to develop its own systems. The users, despite the demanding jobs and work hours, committed themselves to providing the necessary detailed expertise required to make the system a useful, helpful tool. Their efforts were an essential contribution to the projects success. Most importantly, each organization recognized and respected the benefits each group had to offer and worked together to get the job done right.

The development of human resources is a critical aspect of a projects success. The training program made available to the developers was essential to their effectiveness. The support provided by a consultant expert in the new technology was also a substantial help.

B. Prototyping Enables Systems to Exceed "Pre-Defined" Functional Requirements

Although the five increments of NALCOMIS/III were defined in the Functional Design documents, rapid prototyping with a 4GL has enabled some increments to exceed

intended functionality. For example, squadron work centers were not scheduled to be implemented until increment 5; however, the need to include the work centers sooner became evident upon implementation of the Alpha sites. Prototyping with an application generator made it feasible to develop the increased functionality.

C. Prototyping Allows Rapid Recovery from Faulty Design

Prototyping allows for rapid correction of design errors. Under the pressure to quickly deliver an initial product, the NAVMASSO prototyping team made some initial errors in disregarding vocabulary conventions and applying consistent user interface procedures. Even when the mistakes were discovered late during the implementation of test sites, they were corrected in the next release. Another instance of such a correction had to do with the access security. No discretionary access control had been formally defined to prevent unauthorized acts, but this oversight was rectified in the subsequent version. once the problem surfaced during the design of an early version.

D. Existing Operational Test and Evaluation Methodology is Inappropriate for Evolutionary Development

Although there is a requirement for Major Automated Information Systems to successfully complete an Operational Test and Evaluation, the current OT&E strategy does not adapt to the incremental prototyping approach. The hardware used for the prototype was not — by intention — the operational hardware. The deficiencies found with the hardware during OT&E should not be considered as critical to the evaluation of the software being tested and evaluated.

Additionally, prototyped systems may be introduced to the users with partial functionality. These systems should be tested against the currently recognized user requirements instead of preconceived design specifications.

E. Design Documentation Should be Updated to Reflect Evolving System Design

The importance of design documentation in performing maintenance on operational systems cannot be overstated. Maintaining current documentation when designs can change several times a day is impossible. It is probably unnecessary as well since a 4GL specification is largely self defining. Once a design becomes relatively stable, other documentation can more easily be brought up to date. The problem is likely to be substantially relieved in the future through the automatic generation of documentation by such tools as I-CASE products in the future. Until the time when such tools are available, however, documentation of user-driven changes on system functional requirements needs to be updated routinely and later integrated into the OT&E process.

F. Management Must Provide a Proper Environment for Prototyping

The rapid development enabled by INFORMIX/4GL also created some configuration management problems. As errors, modifications, and enhancements were reported back to developers, programmers would use the copy of the software that had been current earlier that day, or the day before, to make the changes. Since modifications to the software could be made so quickly with the 4GL, the developers often found the release they had loaded on their system only hours before was already out of date. Old bugs were reintroduced during functional testing. The developers did understand the problem after a few incidents of this type, but configuration management remained difficult due to the rapid modifications.

No widely accepted standards for programming with a 4GL exist to date. NAVMASSO dealt with the lack of formal rules by establishing the database before programming began. Meetings were held to standardize data elements and variable names. This practice proved a valuable time saver. However, lack of standards in other areas such as backing out of screens and system error messages was a problem for users.

Standard interface principles (e.g., always using the F1 function key to back out of a screen) can and should be established prior to the prototyping process. Testing and debugging need to be performed in a systematic and integrated manner to avoid uncontrolled multiplication of versions. Maintenance should be done on the integrated software and not on the functional components.

G. Application Generators Must be Carefully Selected

When developers choose to build applications with COTS application generators, they should allocate adequate time to evaluate and select the appropriate software/tool for the task. INFORMIX/4GL has been adopted *de facto* in NALCOMIS/III with no formal evaluation. NAVMASSO staff viewed INFORMIX/4GL as a consistent extension of the DBMS module of INFORMIX, which was thoroughly evaluated. Although INFORMIX/4GL has proven suitable for this application, a formal evaluation and selection process would have been required to ensure the appropriateness of the 4GL. Some evaluation methods for selecting COTS are suggested in the Appendix.

H. Software Development Contract Characteristics Should be Reevaluated

Both contracts awarded in the NALCOMIS program were five-year, lowest-bidder, best-effort contracts. DoD outsources the development of many large MIS projects that could require more than five years. Replacing contractors part way through development is a risky practice. Lowest-bidder contracts are acceptable when every detail of the task at hand can be stipulated in the contract. MIS development contracts cannot usually be so clearly defined. Rather than trying to anticipate every detail and hiring contractors to give their "best effort", a more flexible contract stipulating the final deliverable is appropriate in MIS development situations.

I. Current DoD Hardware Acquisition Regulations Hinder System Development

The hardware acquisition process mandated for DoD purchases of computer system hardware is prone to problems. Great care and time must be spent to develop the Request For Proposals (RFP) to ensure the wording does not inappropriately narrow the field of potential bidders. However, even the most carefully worded RFPs fall subject to protests, tying the acquisition process up in rewrites, negotiations, and legal battles. NALCOMIS/III is no exception. Regulations pertaining to hardware acquisition impede the effective use of the prototyping process.

Appendix. Evaluation Methods for Selecting Commercial-Off-The-Shelf (COTS) Software

In situations where one alternative is not clearly superior to the others, we must evaluate the options for the most appropriate choice. If the alternatives are all equal, the selection can be random. However, that is rarely the case. There are three criteria that will guide our decision making process. They are maximum available budget, minimum performance requirement, and maximum effectiveness/cost ratio. Considering budget allowance and performance criteria in isolation may lead us to choose the wrong alternatives. Considering the maximum payoff per unit of investment will generally lead us to a more acceptable solution. The following discussion will explain some methods that consider this criterion.²³

1. Net Value Analysis

One method of rating the alternatives is to estimate the dollar values for each of the criteria. There are two levels of analysis to estimate these values. The first method is to use a rough estimate of the value of that criterion. Although this assessment is quick, it may be superficial and hard to justify. A more detailed assessment can be acquired by analyzing the number of times a particular attribute will be needed and the amount and value of the resources saved by the quality. Although this more thoughtful analysis will result in a more defensible estimate, the required effort is more significant than the former alternative. How much time spent analyzing the alternatives should depend on the cost difference of the options to be considered.

²³Boehm, 1981.

2. Figure-of-Merit Analysis (Weighted Sum Technique)

The Figure-of-Merit technique is an attempt to assign a dimensionless value to each option. The option of choice will be the alternative with the higher figure-of-merit. The following steps depict the weighted sum technique.

- Assign a set of weights to the criteria that will determine a ranking of importance.
- Rank each of the criterion on how well the alternative satisfies the criterion. (Usually a value from 0 to 10.)
- Multiply the rating by the weight.
- Sum the weighted ratings for each of the alternatives.

Although this approach allows us to stress the criteria which are most influential, it is very sensitive to the weights and ratings we assign.

3. Delivered System Capabilities (DSC) Figure-of-Merit

$$DSC = SC * DC * AV$$

where:

- System capability (SC) is defined as a hierarchical weighted sum of individual criterion ratings (equation)
- Delivered capacity (DC) is defined as the actual computer capacity which can be used to provide the desired capabilities.
- Availability (AV) is defined as the fraction of time that the computer system is available to deliver computer capacity to perform the functions. Thus AV excludes time spent on preventive maintenance or system down time.

The DSC approach considers the multiplicative effects of delivered capacity and availability, whereas the weighted sum approach considers them additive.²⁴

²⁴There exists other Multiple Criteria Decision Methods (MCDM) that could be used for product selection.

Glossary of Terms

4GL - Fourth Generation Language

AIMD - Aircraft Intermediate Maintenance Department

APPLICATION GENERATOR - software enabling the creation of application programs based on user provided requirements.

AV-3M - Aviation Maintenance and Material Management System

CASE - Computer Aided Software Engineering

COBOL - Common Business Oriented Language

COTS - Commercial Off The Shelf

DBMS (DATABASE MANAGEMENT SYSTEM) - A set of programs that are used to define, process, and administrator the data base and its applications.

DoD - Department of Defense

END-USER - A collective term used for anyone who uses data and applications to provide information.

FDRD - Functional Design Requirements Document

FUNCTIONAL AREA - Any area within an organization that has a definable set of tasks.

HARDWARE/SOFTWARE ARCHITECTURE - A framework to provide the processing power needed to run applications that will generate and distribute information.

INFORMATION SYSTEM - Activities and resources concerned with the creation, gathering, manipulation, classification, storage and transmission of elements of information.

I-CASE - Integrated Computer Aided Software Engineering

IMA - Intermediate Maintenance Activity

INFORMATION DOMAIN - refers to that data which are relevant to functions of a MIS.

MAG - Marine Corp Aircraft Group

MAISRC - Major Automated Information System Review Council

MIS - Management Information System

NALCOMIS - Naval Aviation Logistics Command Management Information System

NAMP - Naval Aviation Maintenance Program

NAVMASSO - Navy Management Systems Support Office

NRMM - NALCOMIS Repairables Management Module

OMA - Organizational Maintenance Activity

OT&E - Operational Test and Evaluation is the process of verifying software meets all specified requirements. This process is required for MAISRC Milestone 3.

PMA - Project Manager Air

PROTOTYPING - The cyclical process of developing working models of software.

SNAP - Shipboard Non-Tactical ADP Program

SSC - Supply Support Center

TQM - Total Quality Management also referred to as Total Quality Leadership.

WATERFALL MODEL - a sequential, structured software development methodology

References

- Allen, Ronald T., *NALCOMIS/OMA: Functional Considerations for Automating Organizational Maintenance Activities*, Master's Thesis, Naval Postgraduate School, March 1988.
- Boehm, Barry W., *Software Engineering Economics*, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1981.
- Emery, James C. and McCaffrey, Martin J., "Ada and Management Information Systems: Policy Issues Concerning Programming Language Options for the Department of Defense", Department of Administrative Sciences, Naval Postgraduate School, Monterey, California, June 1991.
- Navy Data Automation Management Practices and Procedures, *Systems Decisions*, NAVDAC PUB 24.2.
- Navy Management Systems Support Office, "Functional Description for NALCOMIS/III", NAVMASSO Document J-004 FD-002B, 1 April 1992.
- Office of Technical Assistance, *NALCOMIS OMA Benefit Analysis Plan*, 90056-03-NAVY, December 1991.
- Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, Second Edition, McGraw-Hill Book Company, 1987.
- Whitten, Bentley and Barlow, *Systems Analysis and Design Methods*, Second Edition, Irwin, Homewood, Illinois, 1989.
- Wojtkowski, W. Gregory and Wita, *Applications Software Programming with Fourth-Generation Languages*, boyd & fraser publishing company, 1990.

Bibliography

- Alavi, Maryam, et al., "Strategies for End-User Computing: An Integrative Framework", *Journal of Management Information Systems*, 4, 3, Winter 1987-1988.
- Buckler, Grant, "Users Take Note: Once the Plunge is Made, You're Locked in", *Computing Canada*, February 14, 1991.
- Chartley, Steve, "Tackling the Application Software Crunch: 4GLs in a Client-Server Environment a Good Way To Go", *Computing Canada*, November 22, 1990.
- Desmond, John, "On Living 4GLs, GUI Training, UI March", *Software Magazine*, April 1991.
- Dowling, Richard, "Application Generators are Forcing Wise Use of Reusable Code", *Computing Canada*, February 14, 1991.
- Golick, Jerry, "Time to Return to Basics", *Computing Canada*, March 14, 1991.
- Hanna, Mary Alice, "Prototyping Helps Users Get Design Satisfaction", *Software Magazine*, April 1991.
- Howard, Keith, "4GL vs. 3GL — A Debate Rages without Meaning," *Computerworld*, December 3, 1990.
- Pliskin, Nava, and Shoval, Peretz, "End-User Prototyping: Sophisticated Users Supporting System Development", *Data Base*, Summer 1987.
- Schaffer, Evan and Wolf, Mike, "The Next Generation", *UNIX Review*, March 1991.
- Tessier, Douglas, "MIS's Handling of 4GLs Mixture of Struggles and Fatal Flaws", *Computing Canada*, February 14, 1991.
- Kador, John, "4GLs from the IS Manager's Perspective", *System Builder*, August/September 1990.

Livingston, Denn, "How Integrators Choose 4GLs", *Systems Integration*, July 1991.

Miles, J.B., "4GLs and CASE," *Government Computer News*, January 7, 1991.

"The Nature of 4GLs", *Systems Builder*, June/July 1989.

Distribution List

<u>Agency</u>	<u>No. of copies</u>
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Office of Research Administration Code 08 Naval Postgraduate School Monterey, CA 93943	1
Library, Center for Naval Analyses 4401 Ford Avenue Alexandria, VA 22302-0268	1
Department of Administrative Sciences Library Code AS Naval Postgraduate School Monterey, CA 93943	1
Director of Information OASI (C3I) Washington, D.C. 20301-3040	10
Tung X. Bui Code AS/Bd Naval Postgraduate School Monterey, CA 93943	10