

2627

NRL Report 7909

# The MUDD Report: A Case Study of Navy Software Development Practices

DAVID M. WEISS

*Information Systems Staff  
Communications Sciences Division*

May 21, 1975



NAVAL RESEARCH LABORATORY  
Washington, D.C.

Approved for public release; distribution unlimited. DDC-A

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL REPORT 7909	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE MUDD REPORT: A CASE STUDY OF NAVY SOFTWARE DEVELOPMENT PRACTICES		5. TYPE OF REPORT & PERIOD COVERED An interim report on a continuing NRL problem.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) David M. Weiss		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, D.C. 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NRL Problem B02-18 Project XF21-241-021-K211
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Naval Electronics Systems Command Washington, D.C. 20360		12. REPORT DATE May 21, 1975
		13. NUMBER OF PAGES 32
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software Software engineering Software development Programming		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The MUDD report is a study of Navy software-development practices which is based on a series of interviews with those responsible for the development of Navy systems. The study chronicles the development of a fictional system with requirements typical of Navy tactical systems currently operational or under development. A history of the decisions made during the development of the system is first given. Following the history is an analysis of the impact of each decision on the software developed for the system and on the life-cycle of the software. Finally a set of recommendations for avoiding the pitfalls described in the report is given. The (Continued)		

20. (Continued)

recommendations are designed to assist Navy program managers responsible for software development.

## CONTENTS

PREFACE .....	iv
GLOSSARY .....	1
INTRODUCTION .....	2
HISTORY .....	2
Need for MUDD .....	3
Initial Operational Requirements .....	4
Specific Operational Requirements .....	5
Initial Operating Capability of MUDD .....	7
Development .....	8
Critical Design Reviews .....	9
DUMS Development .....	10
DUMS Coding .....	10
SIPS Development .....	11
SIPS Coding .....	12
MUDD Integration .....	13
MUDD Test and Evaluation .....	13
Final Operational Capability of MUDD .....	15
ANALYSIS .....	15
Consequences of System Configuration Choice .....	15
External Interfaces .....	17
System Development Split .....	17
Coordination with Users and Maintainers .....	18
Choice of Computer, Language, and Support Software .....	18
Critical Design Reviews .....	19
Simulation .....	20
Documentation .....	20
Development Procedures .....	21
DUMS Hardware Change .....	23
MUDD Integration Procedures .....	23
MUDD Test and Evaluation Procedures .....	23
MUDD Maintenance .....	24
CONCLUSIONS .....	25
RECOMMENDATIONS .....	25
ACKNOWLEDGMENTS .....	28

## PREFACE

This report is one result of a year-long investigation into Navy software problems. During this investigation the author talked to many people associated with Navy software development. All of these people, from contracting personnel to program managers to programmers, were keenly interested in finding ways to improve software quality. Although it is not yet possible to give an algorithm for producing reliable software, many mistakes can be avoided by staying aware of problems encountered in the past. The purpose here is to describe some of these problems in a context familiar to Navy software developers in the hope that they can recognize and avoid similar errors in the future.

This report chronicles the development of a mythical software system and describes where and how the developers went awry. The report is based on more than 30 interviews with people responsible for development of various kinds of Navy software in more than ten Navy activities. The pitfalls described typify problems which actually occurred in software development efforts, but all persons and situations described in this report are fictional.

The report is organized into five sections. The first section is a brief introduction. The second section is a chronicle of system development. The reader is urged to try to estimate the effect of the decisions described as he reads about them. The third section is an analysis of the mistakes made by the people who developed the system. The fourth section contains some conclusions about the system development under discussion. The fifth section contains some recommendations which will be helpful to program managers in avoiding the pitfalls described in the second and third sections.

**THE MUDD REPORT:  
A CASE STUDY OF NAVY SOFTWARE DEVELOPMENT PRACTICES**

**GLOSSARY**

ANSI	American National Standards Institute
ASWCCS	Antisubmarine Warfare Command and Control System
CDBDD	Common Data Base Design Document
CDR	Critical Design Review
CONTAC	Consolidation of Tactical Data
CPDS	Computer Program Design Specification
CPPS	Computer Program Performance Specification
CPTP	Computer Program Test Plan
CPTPR	Computer Program Test Procedures
CSDD	Computer Subprogram Design Document
DROWN	Defense Related Ocean Warning Network
DUMS	Data Use and Maintenance System
FOC	Final Operational Capability
FTAC	Fleet Tactical Analysis Center
FTIC	Fleet Tactical Intelligence Center
FTOE	Fleet Tactical Operational Evaluation
FTOSA	Fleet Tactical Operational Support Agency
GECOS	General Comprehensive Operating System
IOC	Initial Operating Capability
JCS	Joint Chiefs of Staff

DAVID M. WEISS

MUDD	Multisource Unified Data Distribution
NERL	Naval Electronics Research Laboratory
NOISY	Naval Office of Intelligence Systems
NPA	Naval Programming Activity
ODORS	Office for the Determination of Ocean Reconnaissance Strategy
OSIS	Ocean Surveillance Information System
PMAS	Program Modification Authorization System
PTA	Proposed Technical Approach
SCS	Super Computing Systems
SDS	Ship Data Source
SIPS	Ships Information Processing System
SOR	Specific Operational Requirements
TSOR	Temporary Specific Operational Requirements
WWMCCS	Worldwide Military Command and Control System

**INTRODUCTION**

This report is an analysis of the major problems encountered in developing the Multi-source Unified Data Distribution (MUDD) system. The emphasis is on the difficulties which caused schedule slippage and cost overruns. It is not the intention to provide here a full-scale evaluation of the system but rather to engender understanding of the pitfalls that can occur during the course of a Navy software development effort.

**HISTORY**

The MUDD effort began in the late 1960's. At that time a number of manual and semiautomated data collection and distribution systems existed. The purpose of these systems was to maintain a data base of items of tactical and intelligence interest. The usual procedure for obtaining an item of information from these systems (for example, the number of U.S. intelligence-gathering ships in the Atlantic during October 1965) was as follows:

1. Find someone with a list of all such systems and the people who maintained them;
2. Call each of these people to discover whether their data base had a list of U.S. intelligence gathering ships in the Atlantic;
3. Get them to extract the necessary data from their data base;
4. Compare all lists so obtained and resolve contradictions by some (usually ad hoc) method.

People with unusual fortitude sometimes repeated steps 3 and 4 several times.

### Need for MUDD

The events leading to the birth of the MUDD system occurred one summer when a submarine was reported missing in an area thought to be free of hostile forces. In the course of reacting to this report, the disposition of all forces, U.S. and otherwise, in the area of interest was requested. In addition, because of unreliable communications in the area caused by weather conditions, a request was made for the location of all ships in nearby areas that might serve as communication relays. Retrieval of the additional information took much too long. When a communication link into the area was finally established, all the procedures associated with command and control, such as briefings, transmission of tactical data, and coordination of search plans, had to be accomplished. There was much confusion and further delay. If the submarine had been under attack or in serious trouble, by the time command and control functions were established and operating, it would long have been lost.

Later investigations of this incident disclosed other cases in which retrieval and dissemination of data was delayed past the time when it was needed. A notorious example was the request made by Rear Adm. Sealey concerning the number of intelligence ships used in a certain area during the previous year. Several weeks were required to obtain the requested data. The admiral's staff found that of the four systems which had the necessary information, one was a manual system and could possibly retrieve the data from deep storage in 6 months, one automated system was down for 2 months while a new computer was being installed, with the estimated conversion time to the new computer for the necessary programs being 6 months after installation, another automated system could provide the information as soon as their lead programmer recovered from a nervous breakdown and wrote the programs, in an estimated time of 3 months and yet another automated system offered to provide the data within 2 weeks but accidentally erased the primary tape containing the information while debugging the necessary program, and the backup tape proved to be unreadable.

After the investigation, the Joint Chiefs of Staff (JCS) directed Rear Adm. Sealey to find means for quickly and efficiently retrieving and disseminating tactical data. The admiral then directed that a committee be formed to consider ways of consolidating the collection and distribution of tactical information. This committee became known as the Consolidation of Tactical Data (CONTAC) committee.

The CONTAC committee initially met in November 1968 and was composed of representatives from the Fleet Tactical Intelligence Center (FTIC), the Fleet Tactical Analysis Center (FTAC), and Fleet Tactical Operational Evaluation (FTOE).

### Initial Operational Requirements

After several months of study the CONTAC committee produced a Temporary Specific Operational Requirements (TSOR) document for MUDD. A part of a transcription of a tape recording of the meeting of 18 February 1969 of the Committee, at which an outline for the TSOR was approved, is as follows:

*Capt. J. McGork (FTOE):* Let me review the reasoning behind Ed's [E. Wood] presentation to see if I understand it. Our ships collect an enormous amount of tactical information on a continuing basis. Most of this information is either lost or takes so long to process that it's worthless by the time we see it. Why not install an information-gathering facility on our ships which can communicate in a timely way with a central land-based computer? This computer can keep all that tactical data stored away, cross-index it, and produce it on demand. The information-gathering facility on each ship can be the same, since they're all getting the same kind of information. We could install a standard computer, like that new UYK-7, with a standard program in it to collect information. The information stored on the ship can be periodically transmitted to the computer on the shore, and vice versa. Since we're standardizing all the computers and their programs, we can keep development costs down. The only difference will be in the shore-based computer, which can be the same as the others but needs a different program. Finally, all aspects of gathering and disseminating the information will be under our control, and we can change them as we like. Now does that summarize the case pretty well?

*Mr. E. Wood (FTAC):* Yes sir, that's a good summary.

*Mr. A. Corde (FTOE):* I'd like to point out, Captain, that although conceptually the programs used to gather data on board the ships will be the same, there may be some variation in the programs from ship to ship because the ships themselves are not identical. This might be a problem, but we think we know how to handle it.

*Capt. McGork (FTOE):* Well, wouldn't the differences in the program be minor?

*Mr. Corde (FTOE):* Yes, probably.

*Capt. McGork (FTOE):* I don't see how that would be a real problem then.

*Cdr. T. Sharp (FTIC):* I don't think we should ignore the intelligence aspects here. Why restrict the material collected to tactical information? I think the system should deal with both tactical and intelligence information.

*Capt. McGork (FTOE):* Good point. When the final TSOR is written, let's emphasize both the tactical and intelligence aspects of this thing. We may be able to open more funding channels that way.

By the end of March 1969 the TSOR describing MUDD was written. It depicted a system in which ships at sea gather various kinds of tactical and intelligence data, store the data in an on-board computer, and transmit it periodically to a shore-based computer. The shore-based computer maintains the data it receives in a large data base and provides facilities for data analysis and transmittal back to the ships. The computer simultaneously operates in batch, time-shared, and communication modes so that it can output data on a terminal display for the benefit of intelligence analysts or a line printer and so that it can maintain communications with its data sources.

While the TSOR was being written, the Navy intelligence community heard of the plans for the proposed MUDD system. Realizing the potential of the MUDD system as a data source, intelligence was interested in contributing to the development of MUDD and using the system when it became operational.

### Specific Operational Requirements

Between March and June 1969 the CONTAC committee met several more times and was expanded to include members of the Navy intelligence community. Representatives of the Naval Office of Intelligence Systems (NOISY), the Office for the Determination of Ocean Reconnaissance Strategy (ODORS), and the Defense Related Ocean Warning Network (DROWN) were added to the committee. Based on the influence of the additional committee members, the original MUDD concept was expanded and Specific Operational Requirements (SOR), along with a Proposal Technical Approach (PTA), were written. The MUDD concept, as expressed in the SOR, is to provide a system for rapid collection, correlation, and distribution of tactical and intelligence data gathered by various classes of ships at sea. Each ship designated as a MUDD source establishes a data base of tactical and intelligence data. This data base is transmitted either directly or via a relay station (perhaps a satellite) to a land-based computer. The land-based computer maintains a central data base containing all data received from ships, provides a facility for correlating and analyzing the data, and transmits relevant data back to the ships and to various interested intelligence systems. The shipborne part of the system described in the SOR was unchanged from the TSOR. The land-based system now had the additional burden of communicating with intelligence systems, some still in the predevelopment stage, such as OSIS and ASWCCS.

DAVID M. WEISS

Before disbanding, the CONTAC committee had one major decision left to make: the assignment of responsibility for the development of MUDD. The committee debated this question over the course of several months and finally came to an impasse, as indicated by the following excerpt from the minutes of an October 1969 meeting which summarizes the nature of the dispute:

Capt. McGork (FTOE) presented a summary of the discussion to date on the question of system development responsibility. He mentioned that each activity represented here has made a case for obtaining development responsibility. FTOE has the most experience in fleet operations and their evaluation. FTIC is the only agency represented here that combines fleet operational and intelligence experience. FTAC argues that the most important aspect of the system is tactical and therefore it should get the responsibility. NOISY has the most experience with intelligence systems and their development. DROWN claims the highest priority based on national defense and its needs. ODORS believes it has the best overall grasp of the relationship between fleet strategic operations and intelligence needs. Mr. Smith (NOISY), on behalf of Mr. Jones (ODORS), Mr. Brown (DROWN), and himself, suggested a compromise whereby the intelligence activities would assume joint responsibility for system development. This was rejected by the fleet activities. Cdr. Sharp (FTIC) then suggested, on behalf of the fleet activities, that a coalition of the fleet activities do the development. This was rejected by the intelligence activities. Mr. Wood (FTAC) proposed that the question, along with the various alternative solutions suggested be submitted to Rear Adm. Sealey for arbitration. The proposal was accepted, and Mr. Wood was directed to write a memo to the admiral explaining the situation.

Rear Adm. Sealey met with the CONTAC committee on 4 December 1969, discussed the situation with them, and informed them of the decision he had made. A transcription of a portion of that meeting is as follows:

*Rear Adm. Sealey:* Gentlemen, this reminds me of one of the problems facing Solomon, and I plan to solve it in the same way he proposed to solve that one. You seem to have divided yourselves into two groups—the fleet activities and the intelligence activities. This system also divides naturally into two parts: an information gathering part and an information distribution part. OK. To me that means that one of the fleet activities should assume responsibility for the information gathering part, which is mostly concerned with fleet operations, and that one of the intelligence activities should assume responsibility for the information distribution part. FTOE will develop everything that's needed for gathering information on board ship and transmitting it to the central computer. NOISY will develop everything else. FTIC and FTAC will consult with FTOE, and ODORS and DROWN

will consult with NOISY. FTOE will also take responsibility for coordinating the two efforts. Capt. McGork, I want to see two PTA's, one for each part of this system, within 8 weeks. Any questions?

### Initial Operating Capability of MUDD

Once the question of development responsibility for MUDD was settled, progress through the design stage of the development cycle was rapid. By mid-1970 an initial operating capability (IOC), along with cost and time estimates, had been defined, the computers and programming languages to be used were established, and the agencies to do the software development work were selected. Estimated completion time for the IOC was 18 months. Because of its proposed role as the standard Navy computer, the AN/UYK-7 was chosen for use by the MUDD system for both ship- and shore-based facilities. The Naval Electronics Research Laboratory (NERL) was chosen to develop the software for the shore-based subsystem by NOISY, ODORS, and DROWN because of the extensive experience NERL had in developing intelligence systems. The fleet activities selected the Fleet Tactical Operations Support Agency (FTOSA) to develop the software for the ship-based subsystem. FTOSA at that time was also simultaneously developing several other shipboard systems using AN/UYK-7 computers and was experienced in software development for shipboard computers.

Both NERL and FTOSA decided to use the CMS-2 programming language to develop their subsystems. The basis for the choice of CMS-2 is contained in a memorandum of 15 June 1970 from Lt. Cdr. Swift of FTOSA to Capt. McGork entitled "Selection of a programming language for the ship-based MUDD subsystem," part of which follows:

1. Candidate languages for programming the shipboard MUDD subsystem fall into two categories: assembly language and high-level languages. Examples of possible high-level languages are CMS-2, FORTRAN, COBOL, ALGOL, and PL/I.
2. Assembly language provides the required speed but is expensive in terms of development time and cost and maintenance cost. In addition, assembly-language programs are difficult to modify. For these and other reasons, current programming trends in the naval, commercial, and academic worlds favor high-level languages over assembly languages.
3. High-level languages such as FORTRAN, COBOL, ALGOL, and PL/I provide many advantages in terms of programming and computational power, ease of development and maintenance, modularity, and modifiability. The only high-level-language compilers available for the AN/UYK-7 computer are for CMS-2. Use of any other high-level language would require a high initial investment in funds and time to write a compiler.

4. Until recently CS-1 was the standard Navy tactical programming language. It has now been superseded by CMS-2. The CMS-2 language embodies a high degree of machine independence, contains a definitional facility for high-level constructs, and provides extension capabilities.
5. OPNAVINST 03500.27A of 6 March 1969 designates CMS-2 as the standard high-level language to be employed in the production of operational processor programs for tactical data systems.
6. It is recommended that the CMS-2 language be used for development of the ship-based MUDD subsystem.

## Development

During the time MUDD was under development, the documentation standard Weapons Specification WS-8506, revision 1, was published and established as a Navy standard. The original MUDD specifications did not conform to WS-8506 because the standard did not exist when the specifications were written. Later revisions of the specifications, and all MUDD documentation written after WS-8506 was published, were written to conform to WS-8506. The documents mentioned here are those required by WS-8506.

The first step in the development was to produce the Computer Program Performance Specification (CPPS). Three variations of this document were eventually produced; one version for the entire MUDD system, one for the ship-based subsystem, and one for the shore-based subsystem. Events in the development followed the pattern established by splitting the software development responsibility and producing separate CPPS documents.

Prior to the start of system design, a simulation program for the shore-based system was developed. (The developers of the ship-based system never attempted a simulation.) This program began as a high-level functional simulation to gather statistics on system capability and was initially used to demonstrate the feasibility of using a UYK-7 computer for the shore-based system. An early attempt was made to use this simulator as a system simulator for use in program checkout but was later abandoned because of the effort involved in keeping it up to date.

FTOSA and NERL proceeded to design the ship- and shore-based subsystems independently of each other with occasional coordination meetings to discuss communications between the two subsystems. A Computer Program Design Specification (CPDS), Computer Subprogram Design Document (CSDD), Common Data Base Design Document (CDBDD), Computer Program Test Plan (CPTP), and Computer Program Test Procedures (CPTPR) were produced for each subsystem. A Critical Design Review (CDR) for each subsystem was then held.

Before the CPDS, CSDD, CDBDD, CPTP and CPTPR documents were produced, only the project sponsor and small system design groups at NERL and FTOSA saw the proposed designs. Neither design groups included an operational commander, intelligence analyst, or maintenance programmer.

## Critical Design Reviews

The two CDR's followed essentially the same course with the same results: the two designs were approved and coding began. Only the CDR for the shore-based system is discussed here.

The CDR was held on 31 October 1970, approximately 5 months after definition of the IOC. One copy of the CPPS was delivered to NERL by its software contractor, Super Computing Systems (SCS), on 30 September 1970, 1 month prior to the CDR. One copy each of the CPDS, CSDD, CDBDD, CPTP, and CPTPR was delivered to NERL 2 weeks prior to the review. (NERL then distributed copies of these documents to all CDR attendees and to all potential users of the system.) The total amount of documentation was 723 pages. The program manager found the documents to be quite thorough and detailed. Attending the review were representatives of FTOE, NOISY, ODORS, DROWN, NERL, and SCS. With the exception of the NERL and SCS people, these representatives were only recently introduced to MUDD. The original CONTAC committee members played a role in the MUDD project for 2 years (until September 1970), but by the time of the CDR all of them had been reassigned to other projects. The description of the CDR presented here is taken from an interview with one of the attendees.

The CDR took about 3 hours. We spent the first hour and a half listening to a presentation made by someone from SCS. He talked about the functions of the system, the computer configuration, and all of the software modules. As I recall, there was something like four modules, one for each system function like message processing, display handling, data analysis, and correlation. He went through a flowchart of each module, showing all of its submodules and their purposes. I think he would have gone into more detail, but there seemed to be a lot of questions from the management people on exactly what the system, and each of its modules, was supposed to do. Then he gave us a detailed breakdown of all the milestones for each part of the development cycle. After that he went into the reporting procedures they were going to use, the weekly and monthly meetings, and that kind of thing. After he finished, we had a question-and-answer session, mostly about backup procedures, operator interface, and printout formats. They had samples of every kind of output or input the system could handle. It was a pretty good review. They had detailed answers for everything and covered the whole system pretty thoroughly. By the end of the review, everyone was satisfied with the design.

Since there were no objections to the design at or after the CDR's, work proceeded on both subsystems. By December 1970, 2 months after the CDR's, coding of both subsystems was well started. From this time until total system integration the two software development efforts proceeded independently, except for the communication modules. The only interface between the subsystems was the messages they exchanged; hence this was the only topic the two developers discussed jointly. The history of the two development projects, up to system integration, will be discussed separately here.

## DUMS Development

The shore-based subsystem was known as the Data Use and Maintenance System (DUMS). DUMS was functionally partitioned into four areas: communications, message processing, data base update and maintenance, and on-line user interaction.

Since DUMS was required to handle data of varying security classifications, it was partitioned into two separate subsystems. One subsystem, known as the high-level system, processed all data at the Top Secret level. Data at the Secret level and below was handled by the other subsystem, known as the low-level system. The requirements for the high-level system included the capability for sanitization and transmission of data to the low-level system. By using what were essentially duplicate subsystems, the DUMS developers hoped to prevent, or at least make extremely expensive, unauthorized access to highly sensitive data.

Development activity for DUMS was apportioned as follows: 30% of the time was allowed for design, 40% for coding, and 30% for checkout. A multiprocessor UYK-7 configuration was selected to provide the required computing power. Prior to MUDD few tactical or intelligence systems used multiprocessors. The DUMS designers, aware that memory sharing in computer systems often caused memory utilization problems in the software, took measures to prevent memory conflicts between programs. The design called for all references to shared memory to be processed by a special module.

The development was contracted to SCS and monitored by NERL. Between the CDR and the beginning of test and evaluation the only people aware of the specifics of development progress and the detailed workings of DUMS were the contractor personnel working on the system, the technical montors at NERL, and a few people at NOISY.

## DUMS Coding

Coding of DUMS proceeded smoothly, and coding milestones were easily met. Approximately halfway through coding, a major change occurred in the DUMS system. A hardware change from the UYK-7 to a WWMCCS Honeywell computer was forced on the developers of DUMS. The results of this change were numerous and included the following:

- All code written up to this point had to be discarded because of the lack of a CMS-2 compiler for the Honeywell machine;
- Large parts of the system had to be redesigned for compatibility with GECOS, the Honeywell operating system;
- Programming personnel had to be either retrained or replaced;
- Estimated cost of the shore subsystem was increased by about 50% (part of this increase was due to initial underestimation; experience with system development enabled a more accurate estimate to be made the second time);

- Estimated system completion time was increased by 9 months (as with cost, part of this increase was also due to more accurate estimating).

Coding of DUMS was declared complete within 2 weeks of the revised estimated completion time, about 1 year after coding started. Checkout of the system took far longer than estimated however. The initial estimate for the checkout period was 30% of the development cycle, or about 5 months. The actual checkout period took about 8 months, or about 60% longer than estimated.

Many of the errors found during checkout were concentrated in mathematical sub-routines. Inaccuracies in ship tracking, which were eventually traced to a position-determination routine in which the programmer used  $22/7$  as an approximation for  $\pi$ , is one example.

Another source of problems was the message-processing routines. The formats of incoming and outgoing messages, especially to and from other intelligence systems, were found to be different than expected. Modifying the system to accommodate unexpected changes to message formats was a problem that plagued the developers and continues to plague the maintainers of DUMS.

A third set of problems occurred when new versions of the GECOS operating system and FORTRAN and COBOL compilers were released partway through the checkout period. A number of new bugs appeared when the new versions went into use. One module, for example, had to be rewritten when there was a change in the meaning of the status bits returned to it by the operating system after a certain executive function was invoked. The change was undocumented, unexpected, and difficult to discover. Several weeks were required to find the resultant bug, which suddenly appeared, intermittently, after the new executive version was released. As with the previous problem, program modification to solve this type of problem has been a continual source of annoyance during the lifetime of DUMS.

In June 1972, more than 1-1/2 years after coding started, DUMS was deemed ready for system integration by NERL. The DUMS developers were somewhat surprised to find that the ship-based subsystem was not quite ready for integration at this point. In fact system integration did not start until September 1972, almost 2 years after the critical design reviews.

### SIPS Development

The ship-based subsystem was known as the Ship Information Processing System (SIPS). The strategy used to develop SIPS was dictated by the number of different versions of the subsystem needed. The development of this strategy was explained in an interview with Donald Dosier, FTOSA project manager for the first year of the development effort:

Our approach to developing the system was worked out in a series of meetings we had with the contractor. The problem was that although we had a general design which specified what modules were required

for an SDS [each shipboard system was known as a Ship Data Source], and we had the specific requirements for each ship type that was included in the IOC, we had no plan for going from the general design to a specific version for a specific ship type. We considered two approaches: one was to write a generalized system and then modify it for each version we needed, and the other was to write one specific version for some ship type and then modify it to fit other ship types. The generalized system idea had the advantages that all the common code could be immediately identified and easily changed as functional requirements changed, a baseline system for reference purposes would always exist, and we could control the modification process easily so we would always know exactly what was being modified. The disadvantage was that we would not have a demonstrable system that could actually be installed and tested on a ship for a long time. On the other hand, if we built a specific version to start with, we could install it on a ship and test it while the other versions were being derived from it. The disadvantage was that we might not end up with as much common code as the other way, since the programmers would be free to modify anything they thought needed modification. The final decision rested with me, and frankly I wasn't sure exactly what to do. The contractor finally convinced me that the two approaches really weren't very dissimilar and that if we produced a specific version first, we could always use it as a baseline. So we picked the largest of the ship classes that were included in the IOC and started to write an SDS system for that class on the theory that since it had more functional requirements than any of the other classes, its SDS system would be the most general.

As with DUMS, the details of development of SIPS were known to only a few people, none of whom were to be involved in its use or maintenance.

### SIPS Coding

Progress on the baseline SDS was quite good. By April 1971, approximately 7 months after the CDR, coding was completed and testing had been started. By mid-June the developers had enough confidence in the system to start several parallel efforts to modify the baseline SDS to fit other ship classes. At this point in the development cycle the contractor apparently started having personnel problems. The contractor was originally chosen by a low-bidder procedure which selected the (technically qualified) contractor with the lowest average cost rate per programmer. Discussions with the contracting officer have since revealed that the contractor maintained his low rates by hiring three experienced programmer-designers at high salaries and a number of inexperienced programmers at low salaries. Of the experienced personnel, one left the company and one was transferred at the end of checkout of the baseline SDS. The third remained with the project and was promoted to a managerial position several months after modification of the baseline SDS began. Their positions were all filled by internal promotions, two from within the project and one (the most senior) from elsewhere in the contractor's organization. The average turnover

rate of contractor personnel for the SDS project was quite high during the entire development cycle.

By October 1971, about 1 year after the start of coding, it was apparent that SIPS was far behind schedule. Milestones were slipped by an average of 12 weeks. More programmers were brought onto the project to start other parallel development efforts to modify the baseline SDS system. This was an attempt to have an SDS system ready for each class of ship in the IOC by the start of system integration. As development continued, the number of versions of SDS modules became quite large. There was a different version of every major module in the system for each ship class under development. In addition each module under development for each SDS had several versions corresponding to different progress levels and different sets of requirements, since system requirements were occasionally changed. Finally, each module had its own set of documentation. The proliferation of different versions of modules required the establishment of a strict system of controls over the coding process. A Program Modification Authorization System (PMAS) was installed in December 1971, more than a year after the start of coding. PMAS was a manual system which required a programmer to obtain approval for starting a new version of a module. All requests for new module versions, along with justification and a description of the difference between the baseline module and the new version, were kept in a single book. This book was searched for duplicate module versions before any module modification was approved. Despite these measures, SIPS was not ready for system integration until September 1972, almost 2 years after the CDR.

One more problem, not specific to MUDD, relates to the coding phase of the software development. The monitoring procedures used to assess development progress included a percent-completed estimate for each module in the system on a weekly basis. These estimates were prepared by the programmers and were often wildly inaccurate. Many programmers had a tendency to report a module as 75% complete at the end of coding. Data compiled from other system development efforts typically show that 40% to 50% of the development effort is spent in testing and debugging. Examination of MUDD progress reports shows some modules, on which work was continuing, as 90% complete for as long as 6 weeks.

### MUDD Integration

System integration proceeded smoothly, in fact better than expected. The interface between the two subsystems consisted of the communication link and the messages transmitted over it. The communication programs for each subsystem were checked out by sending test messages from one subsystem to the other. System integration consisted of sending, receiving, and processing these same test messages. Both subsystems used the same test messages during checkout.

### MUDD Test and Evaluation

The MUDD system was ready for test and evaluation by the beginning of December 1972, approximately 2-1/2 years after the IOC was defined and about 1 year after the initial

estimated completion date for the IOC. During the initial stages of test and evaluation it became clear that many bugs were left in the system and that it worked to no one's satisfaction. One problem was that few people could agree on evaluation criteria. As specified in WS-8506, a CPTP and CTPR were written (one for each subsystem). These were written early in the development cycle, however, and requirements and specifications, particularly for the shore-based subsystem, had undergone many changes. In addition the end users of the system (who had not been consulted since the CDR's), now that they had a chance to observe the system in operation, wanted it modified. The end users with the most complaints generally were either intelligence analysts or operational commanders. Both analysts and commanders had similar complaints: They were not receiving the data they needed, and the system was too inflexible for their requirements. For example, an operational commander could easily obtain a list of all hostile ships which entered or left a given area within the last week but could not obtain a list of hostile ships currently in his area. The existence of two independent developers complicated matters. There was much finger-pointing, cursing, and threatening. A compromise arrangement, involving rewrites of the CPTP and CTPR (as specified in WS-8506) for each subsystem, was reached. This created further delay in system delivery.

The IOC for MUDD was delivered and turned over to the Naval Programming Activity (NPA) for maintenance in June 1973, 3 years after initiation of development and 1-1/2 years late.

In the hardware world, maintenance means the prevention and detection of component failure caused by aging and/or physical abuse. Since programs do not age or wear out, maintenance in the software world is often a euphemism for continued test and debug, and modification to meet changing requirements. This was certainly true for the MUDD system. Maintenance was especially difficult and costly because of the large number of modules in the ship subsystem. One example was the time of 3 weeks and the cost of \$3000 to change a conversion constant from six-place to seven-place accuracy. Most of the cost was absorbed by documentation changes and the time needed to ferret out all modules in which the change had to be made. Most modifications were larger than this and involved changes required by the users.

One requested modification was rejected because it involved a change in formats of messages received by the SDS systems. The reason was that all modules within the system that referenced message components did so by using the disk addresses of the components. The proposed change in the format of the messages would have required a change in the way the messages were stored on the disk. The disk addresses used in all modules which referenced messages would then have had to be changed. The only way to be sure of locating all sections of code using these addresses was to read through all the programs in each SDS system. Reading through all the programs was deemed to be too expensive and time consuming just to change a message format. Somewhat later a conversion module was added which allowed the new format at the expense of translation between old and new formats.

The level of effort required for MUDD maintenance, considering programmers and analysts only, started at about 50% of the level of effort during the coding phase of the development cycle. After about 4 months the level of effort dropped to about 30% of the level during coding and has remained there since.

## Final Operational Capability of MUDD

In view of the 100% time overrun and 150% cost overrun incurred by MUDD, the definition of the Final Operating Capability (FOC) was postponed several times. As of the beginning of 1974 the FOC was still not clearly defined, although several sets of requirements had been written for it. Budgetary constraints have prevented the implementation of a set of requirements satisfactory to all agencies involved in the development and use of the system. Much of the wrangling involved has been similar to the arguments over development responsibility in the original CONTAC committee.

## ANALYSIS

It would be desirable to give an algorithm for the proper development of systems such as MUDD and then compare it with the process actually used by the developers. Unfortunately the art of software development has yet to be reduced to an engineering discipline, and no such algorithm is known. The best that can be done is to use hindsight-aided perception to point out the kinds of decisions that adversely affected MUDD development. Some of these decisions were external to MUDD, some were the best of several bad alternatives, and all seemed quite logical to the people who made them at the time they were made. Each contributed in some measure to increased cost and decreased performance of the MUDD system.

This section analyzes the effect on MUDD software of various development decisions. Most MUDD software problems were not the result of a single decision; hence references to the same problem appear in several different places. Similarly, references to suggested remedies appear in several different places.

### Consequence of System Configuration Choice

A major consequence of deciding to install MUDD systems on ships of different classes is that the software will vary somewhat from ship to ship. This is because different ships have different equipment, have different onboard computer configurations, and collect somewhat different data. There are two ways of developing software under these conditions. One way is to develop one set of code for all ships and never vary it. The other way is to develop modules which support different requirements. All modules which support the requirements of a particular ship can then be combined to produce the system for that ship.

If one set of code is developed for all ships, different ship configurations can be represented by different sets of initialization data, and there is then only one set of code to maintain for all ships. The feasibility of this approach depends on how great the difference in shipboard equipment configurations are. (Talks with MUDD maintenance programmers indicate that this technique was not feasible for MUDD because of the large variation in shipboard equipment.) However, it seems that this sort of approach was never even considered by the designers, even though it had been successfully used in other programs, such as the fire-control software for the Talos and Terrier missiles.

If the modular approach is used, then unless the software is organized in such a way as to minimize the number of modules that change from one configuration to another, large amounts of software have to be customized for each ship. The results are increased development costs because of the extra software that must be written, inability of project management to keep track of software development, voluminous amounts of documentation to chronicle differences in different ships' systems, and a nightmare job of testing, debugging, evaluating, and modifying the software. The software design for the MUDD system becomes doubly difficult: the designers must take into account not only the effectiveness of the design for any given ship, but must also consider the effect of the design on total system development. In other words a good design for a one-ship MUDD system may be a poor design in terms of the amount of code that must be changed to fit it into a multiship system. The key consideration in the software design can be expressed as follows: Each module in the system must provide ways of using some facility of the system without revealing the method used to implement that facility. This idea is known as information hiding. A system designed with information hiding in mind allows one module to be modified without causing modifications in other modules. Such a design may be quite difficult to accomplish and may seem unnatural to a designer who considers the most important considerations to be efficiency in time and space. One example of the use of information hiding is that the module which accepts input data need not know the details of how and where this data is stored in the data base. Otherwise, when the data base format is changed, the input module must be changed also.

In view of their problems in producing SDS systems, the most important design question confronting the SIPS designers was how to partition the system in such a way that it would be easy to modify. This question was never asked, let alone answered. The interview with the FTOSA project manager concerning development strategy shows a lack of awareness of the key issues involved in partitioning software.

Rigorous application of the information-hiding principle in conjunction with a modular development approach could be expected to have yielded the following benefits to the ship-based subsystem developers.

- Production of different SDS systems could proceed on a module-by-module rather than system-by-system basis; that is, a programmer modifying a module would not have to worry about the effects on other modules of his changes.
- Programmers need not understand the entire system before starting work on any particular module. The effects of high programmer turnover rates are thereby minimized.
- A library of modules sharable among SDS systems could be established.
- Design review documents would be readable, and design decisions could be evaluated.

The designers of the ship-based subsystem did in fact use a modularized design but did not design or implement the modules according to the information-hiding principle. Most modules in each SDS system took advantage of specific implementation details of other

modules in the interest of efficiency. The result was that a change in one module set off a chain reaction of changes in other modules, with some of the changed modules not even referencing the module originally changed. In such a situation, adding programmers to a project can be more hindrance than help. The imposition of PMAS was an attempt to control the module explosion which resulted from the poor design technique used. Two results of PMAS were an increase in documentation and a decrease in programmer productivity; programmers waited until approval was granted for each module before coding it. Naturally system development time and cost were both increased.

Another effect of module proliferation manifested itself as difficulty in estimating system progress. Completing a set of modifications to a module yielded little information concerning progress in modifying an SDS system to fit a different ship class because of the chain reaction effect.

As we shall see later, the effects of the multiship configuration decision combined with the bad modularization strategy adopted by the SIPS developers continue to plague the MUDD system.

### External Interfaces

One requirement imposed on MUDD early by the CONTAC committee was that MUDD exchange data with various intelligence systems. As a result the shore-based subsystem was continually plagued by changes in data formats required for communication with other systems such as OSIS and ASWCCS. Data interfaces were defined by negotiation between representatives of the systems. Seemingly arbitrary interface changes occurred often, sometimes unheralded by documentation and in spite of negotiations. There were (and are) no standard data formats for transferring data between computers and no agency for arbitrating disputes over where the responsibility lies for providing various types of data. Software designers must accept the uncertainties inherent in communicating with systems beyond their control as a fact of life and design accordingly. The principle of information hiding may be applied in this situation. If knowledge of the data formats used in communicating with another system are confined to a single module, only that module need be changed when the formats change. The same concept applies to communications with the operating system. Some of the errors most difficult to find in DUMS were caused by operating-system changes. Finding and correcting these errors would have been easier if they had been confined to a single module.

### System Development Split

Another early MUDD decision that had significant effects on the cost of the system was Rear Adm. Sealey's compromise. Both DUMS and SIPS performed many similar functions on the same data, and many opportunities for sharing programs were not exploited. Besides design and coding, many other duplicate efforts could have been avoided by a unified development effort. Examples are critical design reviews and documentation. A more specific example is the communication subsystems. Both the ship- and shore-based subsystems had to transmit and receive messages in the same format. A common design would have

allowed sharing of code and documentation. Not only the development task would have been simplified but also checkout and maintenance.

In addition to software considerations, dividing the developers into tactical and intelligence groups affected the type and format of data received by the end users. Operational commanders using an SDS received data from DUMS, whose developers were mainly interested in the intelligence considerations. The reverse was true for intelligence analysts using DUMS. The effect of this conflict of interest did not manifest itself until test and evaluation, after the software was developed and when it was hardest to modify.

### **Coordination with Users and Maintainers**

Once the software-development phase of a system is declared complete, remaining problems are often left to be worked out between the users and the maintainers of the system. Often, neither of these groups has participated in system development. With one exception the MUDD developers did their best to maintain this tradition. The only opportunity given to end users of MUDD to comment on the system design occurred just prior to the CDR's, when they received a copy of the design documents. The maintainers did not see MUDD until after it was accepted. As a result many changes to the system were requested after it was delivered. Because it took the maintenance programmers time to learn the system, initial modifications were accomplished slowly.

In addition to the preceding problems, partitioning the life-cycle support of software among different activities can prevent the designers of a system from receiving feedback from the users concerning design mistakes made. Later systems designed by the same design group will then have the same design errors.

### **Choice of Computer, Language, and Support Software**

Most of the decisions discussed so far had far-reaching effects on software design, coding, checkout, maintenance, and documentation; most were made without serious consideration of these effects. The choices of computer hardware, programming language, and support software to be used were made with consideration of their effects on software as the prime motivation. Unfortunately all of the choices had adverse software effects. The principle of standardizing the computers to be used was based on the idea of obtaining the capability to transport software and data unchanged from one computer to another. The idea was (and is) a good one. Since the Navy standard computer was a UYK-7, it was the necessary choice for the computer. Unfortunately the available software for UYK-7 computers was limited. The only high-level programming language available for the UYK-7 was CMS-2, a new language for which little support software had been developed. There was no operating system (either time-sharing or batch), text editor, linkage-editor, utility library, or reliable compiler available for CMS-2 on the UYK-7. Any of these would have been invaluable in developing, modifying, and testing the modules for the SDS systems. In addition CMS-2 was not well suited for producing large software systems. For example it was difficult to create and manipulate complex data structures in CMS-2. This may be one reason the disk addresses of message components were known to any program module that needed to

manipulate parts of messages. Furthermore, although CMS-2 was supposed to be the standard Navy tactical programming language, as many as five versions of the language existed when MUDD development started.

Several alternatives were available to the MUDD developers at the time they decided to use CMS-2. The major ones were:

1. Develop a set of good support software for CMS-2 and the UYK-7;
2. Develop a compiler for CMS-2 on a computer with good support software and require that this compiler generate code for the UYK-7;
3. Develop a compiler for ANSI standard FORTRAN or COBOL, ALGOL, or some other "standard" high-level language for the UYK-7. Then do all software development in that language on a computer with good support software. As programs are developed, transport their high-level-language source code to the UYK-7 for use with its compiler.

With the aid of hindsight the viability of some of these approaches can be established. From the point of view of a system developer, alternatives 2 and 3 are least expensive, with 3 somewhat more attractive, since one result will be a compiler for a "standard commercial" language for the UYK-7. All programs which conform to the standards adopted for the language will then be available for use on the UYK-7. The disadvantage of this approach is that the high-level language may be inappropriate for tactical, real-time programming. In that case alternative 2 could be considered. We can estimate the cost of alternative 2 or 3 as less than 3 man-years for development of the compiler. Standard compiler development techniques are well known, and such a compiler could be written in less than 1 year, especially for a language such as ALGOL or FORTRAN.

Coding of the ship-based subsystem took about 8 months longer than expected. The corresponding cost overrun was more than 10 man-years. It is not possible to say how much time would have been saved if alternative 2 or 3 were used, but the difference between the cost overrun and the compiler development cost is more than 7 man-years. Additional benefits could have been gained using alternative 3 when DUMS was forced to a WWMCCS computer. Since WWMCCS supports ANSI standard FORTRAN and COBOL, some code might have been salvaged in the change of computers.

From the point of view of the Navy, alternative 1 is probably the least expensive, since good support software will be reusable by many system-development efforts.

### Critical Design Reviews

The most significant events of the design part of the development cycle were the CDR's. The MUDD CDR's were no more than tutorials on the design of the subsystems and their expected courses of development. This was unfortunate, since the CDR's

represented the last chance for detecting design flaws and estimating potential problems before the start of coding. We can point to three factors that prevented the DUMS CDR from being effective:

- It was impossible for anyone to read and analyze the massive amount of documentation within the 1 week allowed for its review before the CDR.
- The change in upper-level management before the CDR insured that none of the upper-level managers would fully comprehend the MUDD system. These managers turned the CDR's into tutorials to aid their understanding of MUDD.
- The amount of time allotted for the CDR was insufficient. A week might have been adequate but not 3 hours.

The CDR for SIPS suffered from essentially the same problems as the DUMS CDR. Because the CDR's were ineffective, no critical analysis of the design of either MUDD subsystem was ever done. An example of the type of design weakness that should have been detected at the DUMS CDR is the constraint placed on shared memory references. The constraint was imposed to prevent possible memory deadlocks between programs. The results were increased software complexity, increased software size, and increased run time of processes using shared memory. One way of looking at the design is that complexity and efficiency in space and time were traded off in fear of a ghost problem. This tradeoff was unnecessary. Other real-time systems under development at the same time as MUDD solved the problems of sharing memory without imposing any special constraints.

It is significant that FTOSA and NERL were each excluded from the other's CDR, since each would probably have been the best critic of the other's design.

### Simulation

The MUDD software was required to handle data from communication lines and sensors and to deal interactively with humans. Simulators are particularly useful in supporting experimental checkout of this type of software.

A simple simulator was written and used for DUMS, but only to establish that the hardware would meet MUDD needs. The attempt to modify this simulator into a development tool failed because no one was willing to expend the effort needed to keep it up to date. This was unfortunate, since simulation can be a powerful aid in developing and maintaining a reliable system.

### Documentation

Both the development programmers and the maintenance programmers at NPA could have been helped considerably by a good set of documentation. Although the set of documents specified in WS-8506 were produced for each MUDD subsystem, they often contributed to development errors and were not of much help in modifying the programs.

The numerous errors found in mathematical subroutines, such as the use of  $22/7$  for  $\pi$ , appear to be partly the result of subprogram specifications that were not written precisely; hence programmers could not understand all the desired effects of the subprograms they were writing. Another way of viewing this is that all the assumptions made by subprograms which called a particular subprogram were not explicitly specified; that is, programmers were given too much design latitude in a programming task.

Good comprehensible documentation whose structure paralleled the structure of the system would have reduced the learning time for new programmers. This was an important consideration for MUDD because of the high turnover rate among programmers who worked on the system.

The documentation, especially for SIPS, was often out of date. This became more and more true as maintenance proceeded, both because of the expense involved in changing documentation for all of the SDS systems when a change common to all of them was made and because of the patches inserted by shipboard programmers. Even had the documentation been current, it would not have helped significantly. One reason is that the motivations for implementation decisions were rarely discussed. This contributed to the fear-of-deletion syndrome to be mentioned in the maintenance subsection. In addition potentially useful flowcharts were often ambiguous or accompanied by cryptic explanations. A simple example of an ambiguous flowchart is shown in Fig. 1. The result of execution of the algorithm given in this flowchart depends on the direction of shift used in the loop. The documenter apparently felt that this was a trivial detail and specified it nowhere.

The dilemma of providing documentation which accurately reflects the current state of a system is difficult to solve. The solution probably does not lie in establishing a rigid, problem-independent documentation standard.

## Development Procedures

Many of the MUDD development problems were either directly attributable to or compounded by bad software design decisions and external influences on the system. Other factors adversely influenced the development, however. One factor was that the programmers used by the contractors to do the coding either had little programming experience or were simply bad programmers. Another was the high turnover rate of contractor personnel. New programmers entering the project had to go through a learning process before they understood the requirements of their part of the system (if they ever did). A possible explanation of these two factors is that the contractor was specified in a low-bidder procedure which selected the (technically qualified) contractor with the lowest paid programmers and designers. A third factor was the dependence on programmer estimates of completion, expressed as percentages, for monitoring progress. Well-defined, demonstrable milestones would have provided a better measure.

Although coding of DUMS was completed within 2 weeks of the (revised) estimated completion date, checkout took about 3 months longer than expected. This apparently was because the computer was changed after some coding had been done. The experience gained by comparing the actual coding time to the estimated coding time was used to

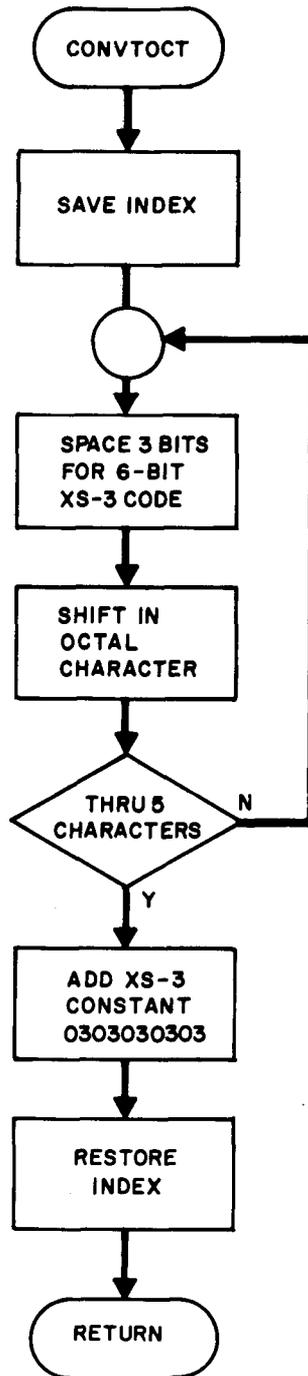


Fig. 1 — Sample MUDD flowchart

forecast accurately the time needed for coding the system on the new computer. No such experience existed for checkout however. Data compiled from other system-development efforts typically show that 40% to 50% of the development effort is spent in testing and debugging. The initial estimate for checkout of DUMS was 5 months, or 30% of the time, and the revised estimate was also 5 months.

### DUMS Hardware Change

The largest disaster that befell the DUMS developers was the hardware change forced upon them. Since the developers had no control over this change, we will not discuss the reason for it. Instead we will discuss why this change was so disastrous.

The largest parts of the cost of the change were associated with redesigning the interface between the software and the computer system and recoding all programs already written. (A side issue associated with the hardware change was whether the cost of the change should be charged to hardware, software, or some combination of the two.) Redesign could not be avoided, but recoding costs associated with retraining of programmers could have been minimized if DUMS had been written in a commercially available high-level language or if a CMS-2 compiler had been available for the new computer. Another effect of using WWMCCS equipment was that the system was now subject to external influences over which the developers had little or no control. New versions of the operating system, compilers, assembler, and other support software were released and old ones discarded, with no approval by the DUMS developers required. As mentioned previously, good software partitioning would have reduced this problem. Some benefits were gleaned from the change of computer. Honeywell support software was somewhat better than UYK-7 support software, and this eased the rewrite task.

### MUDD Integration Procedures

System integration of the ship- and shore-based subsystems proceeded smoothly. One might expect this, since the only point of contact between the two involved communications, and the communication specifications were jointly developed. The remainder of each subsystem was independent of the other and had already undergone checkout prior to system integration. In addition the data used to perform integration tests were mostly the same as the data used to check out the subsystems. This was probably because generation of test data was not easy, partly owing to the lack of a simulator.

### MUDD Test and Evaluation Procedures

It is not surprising that MUDD test and evaluation did not proceed smoothly. The project was conceived in 1969 and did not undergo any evaluation until 1973. During that interval system requirements changed significantly without corresponding changes of the evaluation criteria or the test plan. Moreover, after critical design reviews, the users of the system were not consulted concerning their needs. Finally, as was noted, system integration procedures did not thoroughly test the entire system. Consequently, when

real data were used during operational test and evaluation, many bugs appeared for the first time. Attempts to locate the cause of errors were complicated by lack of trust between the developers. Each contractor tried to fix the blame for programming problems on the other contractor.

Conflicts between intelligence and tactical requirements appeared during test and evaluation. Many requests for system modifications, caused by these conflicts, were made during this time. The example given previously, wherein an operational commander wanted position reports on hostile ships in his area, while the intelligence analyst was mainly interested in the sailing patterns of the ships, is typical. DUMS maintained the sailing patterns and some sighting data on a tape file that was updated once a week in a batch mode. To send tracking data to an SDS on a daily basis required that the data be kept in an online file, which meant either adding it to a new file or fitting it into the data base kept by DUMS. Since there were a number of requests like this, a redesign of the data base was required. Since the format of the data base was known and used by many DUMS routines, the proposed redesign would have been costly and lengthy and was postponed until the "maintenance" stage. The effects of many of the early MUDD decisions, such as splitting the development effort, not continually consulting the users, and not using a good software partitioning strategy, were clearly apparent here.

Because of the many changes to MUDD, both proposed and actual, since its inception, the question of what was an acceptable system was open to much discussion. One possible hindsight-aided solution to this problem is to have had a careful definition of acceptance requirements, with allowances for change, written into development contracts. At least this would have provided firm grounds for acceptance, nonacceptance, or negotiation.

### MUDD Maintenance

The accumulation of many of the problems from the test and evaluation procedure descended on the heads of the maintenance programmers. Since NPA had no hand in designing, developing, testing, or using MUDD, it took the NPA programmers some time to understand the system. Talks with NPA programmers indicate that they were appalled at the job of modifying and debugging MUDD, especially the ship-based subsystem. One fear ever-present in their minds was that they could never detect all the side effects of deleting a section of code. Consequently they never deleted any sections of code. They effected modifications by jumping around sections of code that they suspected were no longer needed. Unfortunately they could not be sure that a section of code was unneeded without an inordinate amount of effort. Of course this made the program more difficult for the next programmer to modify. The amount of code in the system continued (and continues) slowly to grow as modifications are made. The cost of modifying MUDD also continues to grow as it becomes harder and harder to understand any part of the system.

The maintenance programmers were plagued with an additional major problem. Since the lead time for modifying programs was long, shipboard personnel sometimes inserted corrections, known as patches, directly into the shipboard SDS computers. These patches were often passed from one ship to another. Soon a library of patches was circulating

among SDS ships. A maintenance programmer trying to modify or correct an SDS program was unaware of some of the code in the shipboard system; hence his changes often would not work. Maintenance on MUDD was frustrating.

As is true in many Navy systems, the separation of maintenance programmers from software developers prevented the developers from becoming aware of the problems some of their decisions caused. This may lead to a repetition of these problems when later systems are developed.

## CONCLUSIONS

This report shows the kinds of bad decisions, both internal and external to MUDD, which adversely affected MUDD development. Usually the adverse effects were not immediately obvious and generally depended on hidden assumptions that the decision makers were not aware of. Unfortunately consequences of many of these decisions troubled the project throughout its development and continue to trouble it. In a report of this kind it is easy to point out and even deride such decisions. It should be remembered, however, that the MUDD developers were trying to produce a system of a type which had not been produced previously and which used new hardware (the UYK-7) and new software (CMS-2) at the same time. Consequently a model of the development process was not available to the system developers. It is hoped that anyone who undertakes development of a similar system will use MUDD as a source of synthetic experience which will help to avoid similar errors.

## RECOMMENDATIONS

Although there is no known algorithm for proper software development, it is possible to avoid many of the kinds of pitfalls described in this MUDD report. The list of recommendations given below are designed to help those responsible for the creation and support of Navy software systems. Most of the recommendations in one way or another are concerned with interfaces: interfaces between and within systems, interfaces between people, interfaces between the Navy and its contractors, interfaces within the Navy, etc. Each recommendation is as specific as possible, with an attached set of problems, usually drawn from the MUDD report, which the recommendation should help avoid.

- *Unify life-cycle control of software.* Development responsibility for a system should not be split, and maintenance activity should not be independent of development activity. In particular, system maintainers should participate in the development cycle from requirements definitions to delivery. Separation of control over software during its lifetime leads to additional interfaces and inhibits feedback useful for preventing repetition of errors.
- *Require the participation of experienced software engineers in all system decisions.* The effect of early decisions such as determination of system configuration, assignment of development responsibility, and choice of support software on system software is not always obvious.

- *Require the participation of system users in the development cycle from the time requirements are established until the time the system is delivered.* MUDD users never saw the system until operational test and evaluation. Many of the modifications they then requested could not be implemented because the changes were too costly. Changes which are inexpensive and easy at system design time are often extremely expensive and difficult after the software has been written.
- *Write acceptance criteria into software development contracts.* Both the contracting agency and the contractor then have a clear idea of the requirements the system must meet to be accepted. If the criteria are not clearly established in the contract, there may be misunderstanding and a protracted delay for negotiation before the system is delivered.
- *Develop software on a system that provides good support facilities.* If necessary, consider developing support software prior to or in conjunction with system development. Most support software is a good example of sharable software. The DUMS developers were considerably aided by the presence of support software, and the SDS developers were sorely in need of it. Support software includes, among others, assemblers, compilers, operating system, text editors, and management information systems.
- *Design software for maximum compatibility and reusability.* Design decisions that cause one system to be distinguished from another should be delayed as long as possible in the design process. If MUDD had been designed in this way, there would have been little difference between DUMS and SIPS. The differences that existed would have been isolated and easily traceable to a few design decisions.
- *Allocate development time properly among design, coding, and checkout.* Software-development experience indicates that *rough* estimates for these phases are 40% for design, 20% for coding, and 40% for checkout. Some of the variables involved are the nature of the project, the design models available for the project, and the experience of the designers. All developers should keep a file of past experience in this area for future guidance. Since manpower-allocation estimates are based in part on the time estimates for the different phases of development, improper estimation can be quite expensive.
- *List, in advance of design, all areas in which requirements are likely to change.* This can be done at the time requirements are stated and will help the designer to partition the software. The responsibility for identifying all areas where requirement changes are most likely to occur lies with the program manager. He must specify these areas as early as possible to all concerned, including contractors, developing agencies, and system designers.

- *Use state-of-the-art design principles, such as information hiding.* Large systems, such as the ship-based subsystem of MUDD, must be designed using principles that optimize the chances for producing reliable, inexpensive, maintainable software. The resulting design may even seem unnatural to designers accustomed to optimizing for efficiency. Ignorance of information hiding helped produce a MUDD system that was expensive, late, unreliable, and difficult (and sometimes impossible) to improve or maintain. The basic problem in MUDD was that each module took advantage of implementation decisions made in other modules. A change to one module then started a chain reaction of changes in other modules. Naturally, the larger the number of changes required, the lower the probability and the higher the expense of correctly implementing a modification to the system.

The software design should isolate and insulate all areas where requirements are most likely to change. In particular all interfaces with other systems over which the developers and users have no control should be transparent to the rest of the system. Data obtained from other systems can change in format in unpredicted and unheralded ways. Often the only recourse in such situations is to change the module which inputs the data. A change of this nature should not require a change to more than one module. This is an important instance of the need for information hiding.

- *Critical design reviews should be active reviews and not passive tutorials.* Sufficient time must be allowed to read design documents before the review, and the documents must be readable. Alternative design decisions and the reasons for eliminating them should be discussed. In addition no code should be written until the design is approved. The critical design review is the last and most important time to catch errors before coding starts. Once code has been written, any design change involves at least examining all existing code for the impact of the change and may involve discarding and modifying code. System progress is delayed during this process. Consequently the cost of a design change during coding may be 2 or 3 times the cost of the change before coding. The multiplier becomes larger the farther the system progresses in the development cycle.
- *Do not depend on progress reports to know the state of the system.* Progress reports ultimately depend on the programmer's estimate of his progress. Programmers are notoriously optimistic concerning the state of their programs. There is always "just one bug left." Rather than progress reports, milestones can be used as an indication of development progress.
- *Require executable milestones that can be satisfactorily demonstrated.* Without demonstration of soundness, major design decisions may be left untested until it is too late to change them at reasonable cost. Milestones demonstrating system capabilities that will rest on major design decisions should be written into development contracts.

- *Ensure that a proper variety of test data is used.* The differing MUDD experience between system integration and test and evaluation is indicative of some of the problems that arise when a system is incompletely tested. Support software capable of monitoring system tests and reporting on failure and on what code has and has not been tested is now coming into use. Generation of test data can be facilitated by the use of a simulator. Although testing cannot by itself be used to guarantee reliability, it will probably remain for some time to come as the basis for finding errors and inspiring confidence in systems.
- *Maintain current, complete documentation.* Documentation is an often neglected part of software development. In many systems it is done on an after-the-fact basis and rarely updated. This may be because no one knows how to do it properly. Unreliable documentation forces the maintenance programmer to rely on nothing but code reading, a long and tedious process, for his understanding of the system. Well-written documentation will have few redundancies and many cross-references; it will be tailored to suit the system being documented. One sure sign of danger is when coders use unofficial documents and produce the official ones only because of contract requirements.

#### ACKNOWLEDGMENTS

Although this report purports to be the work of a single author, it is his expression of the accumulated software experiences of many people. Thanks are owed to all of those who patiently endured the questions of the author and thereby contributed their time and some part of their experience. Significant contributions were also made by Frank Manola, Dr. David Parnas and Dr. John Shore, all of whom helped clarify many sections of this report. Thanks are also owed to Georgine Spisak, who patiently typed and retyped many early drafts.