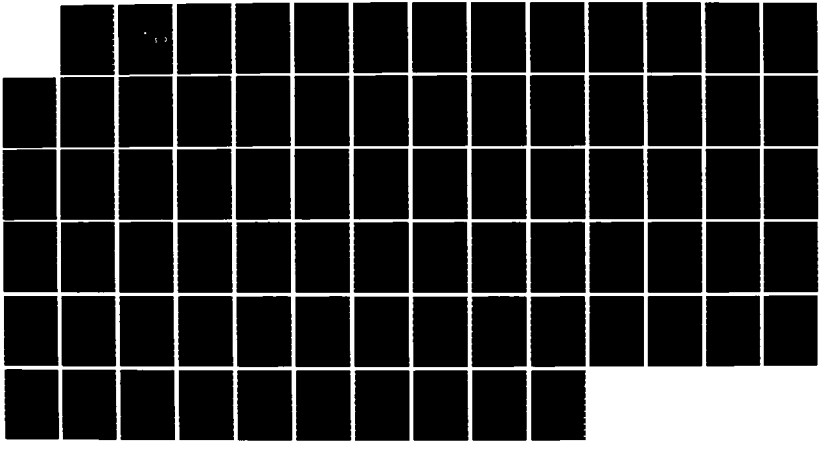
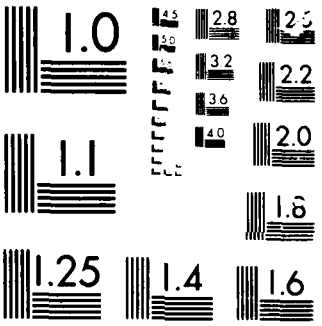
AD-A168 856 AN ANALYSIS OF THE MARITIME TACTICAL SOFTWARE MAINTENANCE MANAGEMENT(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA B W SILVER MAR 86 1/1

UNCLASSIFIED F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.2
2.0
1.8

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN ANALYSIS OF THE MARITIME TACTICAL SOFTWARE MAINTENANCE MANAGEMENT

by

B. Wayne Silver

March 1986

Thesis Co-Advisors: Carl R. Jones
Tung Bui

Approved for public release; distribution is unlimited.

AD-A168856

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School 6b OFFICE SYMBOL (If applicable): Code 54	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

AN ANALYSIS OF THE MARITIME TACTICAL SOFTWARE MAINTENANCE MANAGEMENT

12 PERSONAL AUTHOR(S)
Silver, B. Wayne

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's thesis	FROM _____ TO _____	1986 March	77

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Canadian Forces Maritime Tactical Software Maintenance, Software Maintenance

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

This thesis examines the Canadian Forces Maritime Tactical Software Maintenance organization and its ability to meet the future software maintenance requirements of the fleet. In order to provide new programming managers with a better understanding of Maritime tactical software maintenance, a basic framework for software maintenance is presented, and the major problem areas within the military software maintenance environment are addressed. Software management practices, maintenance techniques and tools, and documentation and testing procedures are all discussed in order to aid the military manager in improving the Maritime software maintenance organization. Based on a better understanding of the life-cycle requirements of software maintenance, and the factors effecting programmer productivity, suggestions for improving the software productivity of the Fleet maintenance organization are discussed. By addressing the problems of the past tactical

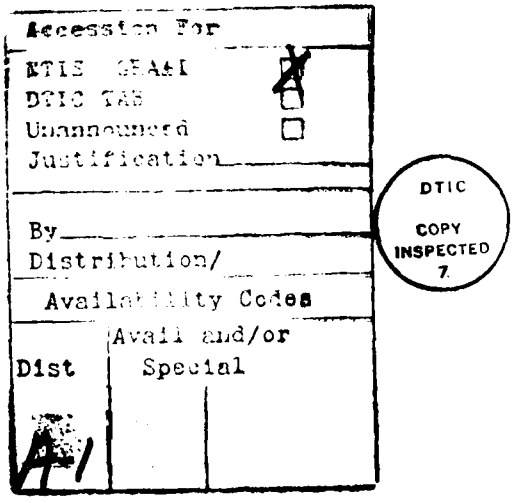
20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Carl R. Jones	22b TELEPHONE (Include Area Code): (408) 646-2767 / 22c OFFICE SYMBOL: Code 54Js

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

SECURITY CLASSIFICATION OF THIS PAGE

19 ABSTRACT (cont'd)

programming environment and understanding the present technology of software maintenance management, it is hoped that the capabilities of the future Fleet Tactical Programming Center will be greatly improved.



Approved for public release; distribution is unlimited.

An Analysis of the Maritime
Tactical Software Maintenance Management

by

B. Wayne Silver
Lieutenant Commander, Canadian Navy
BSc., University of Victoria, 1969

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1986

Author: B. Wayne Silver
B. Wayne Silver

Approved by:
Carl R. Jones, Thesis Advisor

Tung Bui
Tung Bui, Co-Advisor

Willis R. Greer, Jr., Chairman,
Department of Administrative Sciences

Kneale T. Marshall
Kneale T. Marshall,
Dean of Information and Policy Sciences

ABSTRACT

This thesis examines the Canadian Forces Maritime Tactical Software Maintenance organization and its ability to meet the future software maintenance requirements of the fleet. In order to provide new programming managers with a better understanding of Maritime tactical software maintenance, a basic framework for software maintenance is presented, and the major problem areas within the military software maintenance environment are addressed. Software management practices, maintenance techniques and tools, and documentation and testing procedures are all discussed in order to aid the military manager in improving the Maritime software maintenance organization. Based on a better understanding of the life-cycle requirements of software maintenance, and the factors effecting programmer productivity, suggestions for improving the software productivity of the Fleet maintenance organization are discussed. By addressing the problems of the past tactical programming environment and understanding the present technology of software maintenance management, it is hoped that the capabilities of the future Fleet Tactical Programming Center will be greatly improved.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

With the introduction of a new class of computerized destroyers into the Canadian Navy in the late 1980's and the proposed modernization of the present fleet, the amount of embedded shipboard software that will have to be maintained by the Maritime Command will greatly escalate. It is estimated that the presently maintained 200,000 lines of source code will grow to more than two million lines of code [Ref. 1]. Although the Maritime Command does not intend to develop most of these new software programs, the greatest proportion will be maintained by military personnel.

In order to prepare for the challenges that will be posed by the rapid increase in shipboard software, the Canadian National Defence Headquarters (NDHQ), Director General Maritime Engineering and Maintenance (DGMEM), issued a paper [Ref. 1], which identified some of the past problems of the military software life cycle management, proposed solutions to these problems and made recommendations on the methodology, establishment and manning of a Software Support Center. With the Command's support of this paper, Maritime Command Headquarter (MARCOM), issued a report on the establishment of a Fleet Tactical Software Center (FTSC) [Ref. 2], in which it outlined the scope, role, organization, reporting relationships and objectives of a new software maintenance facility.

One of the major conclusions in this MARCOM report was that the determination of a Life Cycle Management Plan to support all identified programs required further detailed study which may or may not be within the capability or resources of existing, available personnel [Ref. 2].

It is not the intentions of this thesis to develop and design the equipment, layout, office and work spaces, and personnel requirements of the FTSC organization, but rather to focus on the management and maintenance of software. It is intended to provide guidance to Maritime Command personnel in order to assist them in improving the software quality and productivity of the proposed fleet software maintenance organization.

There are two primary ingredients to developing a successful software maintenance organization. The first is to recognize problems that are unique to the tactical maintenance environment, and the second is to plan strategies and policies that incorporate maintenance technology, good management practices, and service goals of the organization. It is under this philosophy that this work is organized. Initially the problems within the maintenance environment are examined. Then the various policies, procedures, and techniques available to resolve these problems are presented.

II. OVERVIEW OF THE CANADIAN FORCES MARITIME TACTICAL SOFTWARE MAINTENANCE ENVIRONMENT

A. BACKGROUND

The Canadian Navy began their involvement with embedded software projects in May of 1965. Selected Naval officers were loaned to the United States Navy at San Diego and were employed in writing programs for the initial NTDS systems. Here they were instrumental in the design and writing of a Small Ships Combat Data System (SSCDS) that was trailed by the USN for their destroyers. In late 1965, three of these officers returned to Canada and formed the nucleus of the Naval Hydrofoil Operational Programming Team at Westinghouse in Hamilton, Ontario. Using their SSCDS experience, they produced the first shipboard system for the Canadian Navy. Although it never went to sea, it was successfully used as a training vehicle for military programmers and future Canadian projects.

In 1968, the Candadian government approved the development of a new class of Destroyer (280 class) and a group of seven officers from the Hydrofoil team moved over to form the Combat Control System 280 project team (CCS 280). At the same time the Hydrofoil system was set up as a training establishment at CFB Halifax and called the Maritime Tactical Data Systems (MARTADS). In early 1969, the Underwater

Combat System (UCS 280/257) for both the 280 class destroyers and the older Improved Restigouche (IRE) class destroyers was started.

In 1970, the Combined Support Division (CSD) was established at Halifax with the explicit task of supporting the DDH 280 and IRE projects. In addition to training officers and men as operators and maintainers of the systems, and the computer programme generation, it was tasked to provide Combat Systems Engineering Support to both classes of ships. By 1972 the Combined Support Division had assumed the responsibility for software generation and maintenance for the two major Maritime embedded software project, the CCS 280 and the UCS 280/257.

Although the tactical or operational programmes for both the Hydrofoil and CCS 280 projects were written by Naval officers, the majority of support software, i.e., compilers, I/O handlers, program generation software, etc., have been provided by contracted civilian software firms. Due to the limited manpower resources and software expertise within the service, all major software development was left to outside contractors and the CSD programming section became primarily a software maintenance facility. From 1972 to 1982 a programming staff of one senior programming manager, two program coordinating officers, and approximately eleven officers and three chief petty officers provided the entire software maintenance responsibilities for the Maritime Command.

During this time many modifications to both programmes have taken place, with an average of one operational revision per program per year. In 1975, a complete rewrite of the CCS 280 programmes was undertaken and completed to bring the system into line with the other NATO Navies' LINK 11 standards.

In the early 1980's several new embedded software systems were being developed for the Maritime Command. At that time the CCS and UCS programmes had provided a little over 100,000 lines of source code to be maintained. In 1985, a new Automatic Data Link Plotting System (ADLIPS), a Submarine Fire Control System (SFCS) and a Message Handling System (MHS) were introduced into the fleet, accompanied by their additional 350,000 lines of code. It became obvious that the CSD programming section and its staff of twenty programmers were not going to be able to support this rapid growth in software maintenance responsibilities.

In a 1984 software study [Ref. 1] NDHQ estimated the size (source lines of code) for all Canadian Maritime software programs for the next five years (Fig. 2.1). If, as in the past, the Navy was to support all these systems in-service, the Tactical Support Center would have to maintain approximately 3,125,000 lines of code. As displayed in Figure 2.2, the tactical programming section would grow from supporting six systems to over thirty systems by 1989.

Faced with this explosion in software maintenance, the Maritime Command endorsed the reorganization of their

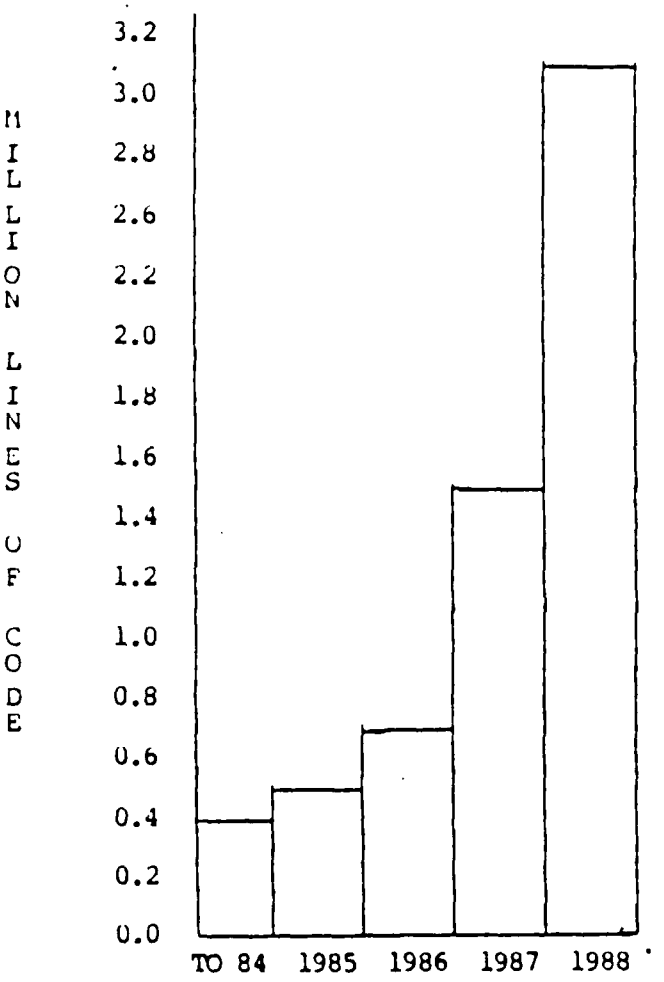


Figure 2.1 Total Estimated Software for Embedded Systems Entering the Maritime Command by 1989

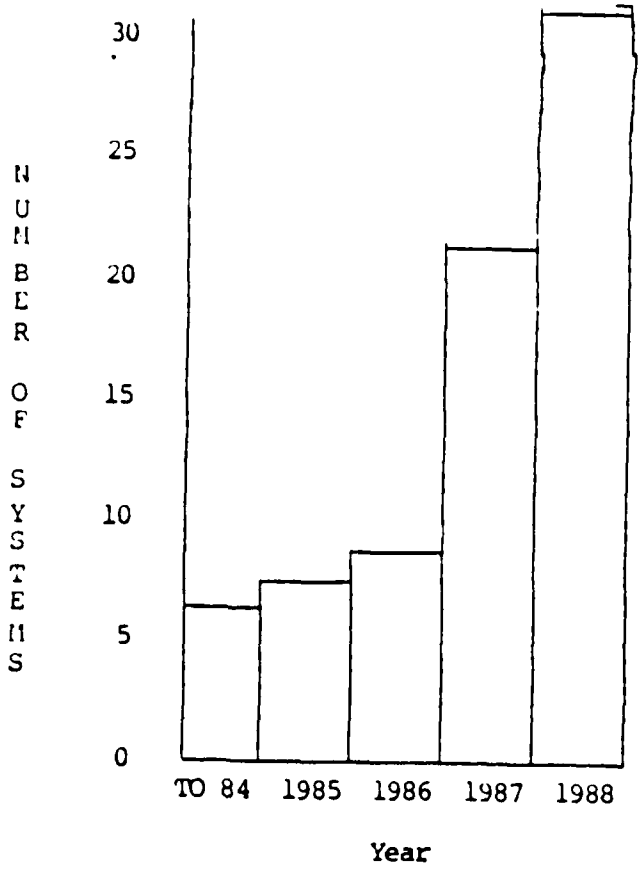


Figure 2.2 Total Number of Embedded Computer Systems Entering the Maritime Command by 1989

programming facilities and the ultimate establishment of a Fleet Tactical Software Cener (FTSC) [Ref. 2].

B. ORGANIZATION

For the first decade of its existence the defined tasks for the programming section of CSD were:

1. Maintenance and issue of CCS 280 and UCS 257/280 operational and diagnostic programs;
2. Evaluation and impact analysis of suggested changes for presentation to the Maritime Operational Shipboard Software Committee Working Group (MOSSCOW);
3. Programmer training;
4. Liaison with Maritime Command, National Defense Headquarters (NDHQ), and NATO organizations, to ensure national and international software compatibility; and
5. Liaison with and support to NDHQ for procurement and evaluation of new systems.

To perform the above duties, the programming section was organized into separate project teams. Each team consisted of:

a project manager	who was usually the senior officer;
an editor	who was the "Second In-Charge", and handled all program configuration management duties;
two/three tactical programmers	who divided up the programming responsibilities for the major operational segments of the system; and
one/two diagnostic programmers	who looked after the diagnostic programming duties.

Each time a new embedded software system was introduced into the fleet, another project team was added to the

organizational chart and, by 1985, the CSD programming section looked similar to Figure 2.3.

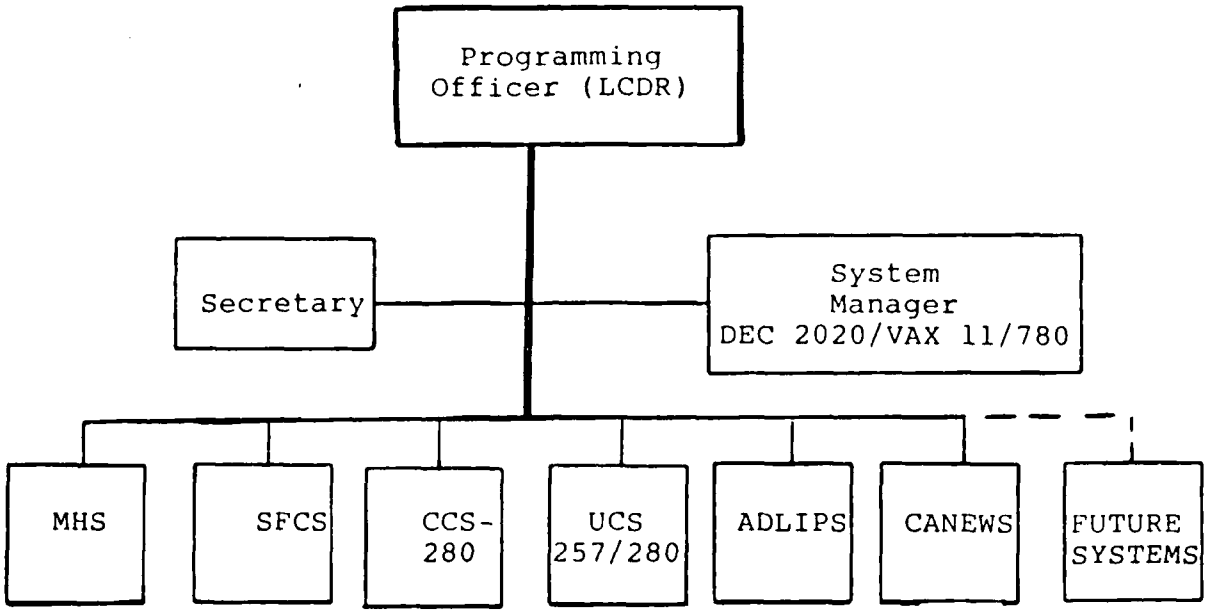


Figure 2.3 CSD Programming Section Organizational Chart 1985

C. PERSONNEL

The Maritime Command presently supports the majority of its tactical software in-service and predominantly by military personnel. Prior to being posted into the programming section, officers and senior non-commissioned officers (NCOs) must successfully complete a nine week Maritime Tactical Programming course. Here they are introduced to the programming environment and are taught to code in an assembler language, ULTRA 16, and the U.S. Navy's higher level language, CSM-2. If the student has demonstrated an acceptable interest and ability in computer programming during this initial

training, he (only one female officer has completed the initial course) can be posted into the section for an average two and a half year posting. Within the section the officer is assigned to one of the major systems and commences an "on-the-job" training course that introduces him to the system he will maintain for the remainder of his posting.

Under this Maritime programmer training system, it has taken approximately a year to produce a programmer capable of working alone. This lengthy training requirement and the continual turnover of personnel have caused a major shortage of qualified, experienced programmers and the resulting lack of continuity and experience level within the support facility.

In addition to the present personnel requirements, the rapid growth in software maintenance responsibilities will be accompanied by an equivalent demand for experienced personnel. It was estimated that by 1988, the present programming staff of thirty military personnel will have to grow to 178 individuals in order to support the additional embedded software systems [Ref. 1]. This overall Naval Shipboard Software study clearly demonstrated the Command's serious shortage of qualified computer personnel and tactical software maintenance expertise. The Command's recent approval of new computer classification sub-specialties and advanced training procedures will assist in retaining its present computer personnel and in providing a viable career

path for those long-term individuals in the computer processing field.

III. FRAMEWORK FOR SOFTWARE MAINTENANCE

A. SOFTWARE MAINTENANCE DEFINITION

The IEEE Standard Glossary of Software Engineering Terminology defines software maintenance as:

> Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment. [Ref. 3]

The U. S. Department of Commerce, National Bureau of Standards defines at as:

> The performance of those activities required to keep a software system operational and responsive after it is accepted and placed into production. [Ref. 4]

The Conference on Software Maintenance - 1985 defines it as:

> The enhancement, restructuring, and correction of software in production use. [Ref. 5]

Although there exists grammatical differences in the above definitions, they all convey that software maintenance is the set of activities which result in changes to the originally accepted (baseline) product. Generally, these changes are made in order to keep the system functioning in an evolving, expanding user and operational environment.

B. CATEGORIES OF SOFTWARE MAINTENANCE

As originally proposed by E. B. Swanson [Ref. 6], maintenance activities are divided into three major categories:

perfective, adaptive, and corrective. These categories are defined as follows [Ref. 7]:

* Perfective Maintenance

  All changes, insertions, deletions, modifications, extensions, and enhancements made to a system to meet the evolving and/or expanding needs of the user. Activities designed to make the code easier to understand and use, such as restructuring or documentation updates are considered to be perfective maintenance as well as optimization to make the code run faster or use storage more efficiently. Estimates indicate that more than 60% of all maintenance effort falls into this category.

* Adaptive Maintenance

  All effort initiated as a result of changes in the environment in which a software system must operate. These environmental changes are normally beyond the control of the software maintainer and consist primarily of changes to the computer hardware, operating system, operating system tools (compilers, utilities, etc.) and terminal devices. Estimates indicate that approximately 20% of all maintenance effort falls into this category.

* Corrective Maintenance

  Changes necessitated by actual errors (included or residual bugs) in a system. It accounts for 20% of all the software maintenance efforts and consists of activities normally considered to be error correction required to keep the system operational.

C. SOFTWARE MAINTENANCE LIFE-CYCLE

As the chairman of the first IEEE Software Maintenance Workshop, December 1983, Norman Schneidewind [Ref. 8], stated that:

> Many managers and software professionals have not appreciated the crucial role of maintenance in determining the quality of software provided to its users. This attitude was largely the result of a failure to recognize that maintenance is an integral part of the entire software development and implementation process.

Software life cycle is defined as the period of time that starts when a software product is conceived and ends when the product is no longer available for use [Ref. 3]. From its initial introduction in 1970 [Ref. 9], the Waterfall model, as illustrated in Fig. 3.1 has been the most widely accepted representation of software life cycle.

Royce's software life cycle has been included in this paper for reference purposes and typically includes the following phases:

1. Requirements phase. The period of time in the software life cycle during which the requirements for a software product, such as the functional and performance capabilities, are defined and documented.

2. Design phase. The period of time in the software life cycle during which the designs for architecture, software components, interfaces, and data are created, documented, and verified to satisfy requirements.

3. Implementation phase. The period of time in the software life cycle during which a software product is created from design documentation and debugged.

4. Test phase. The period of time of the software life cycle during which the components of a software product are evaluated and integrated, and the software product is evaluated to determine whether or not requirements have been satisfied. Within the Waterfall Model, Figure 4.1, this phase is culminated by a verification and validation (V & V) activity whose objective is to uncover as many problems as possible in the product of that phase.

5. Installation and Checkout phase. The period of time in the software life cycle during which a software product is integrated into its operational environment, monitored for satisfactory performance, and modified as necessary to correct problems or to respond to changing requirements.

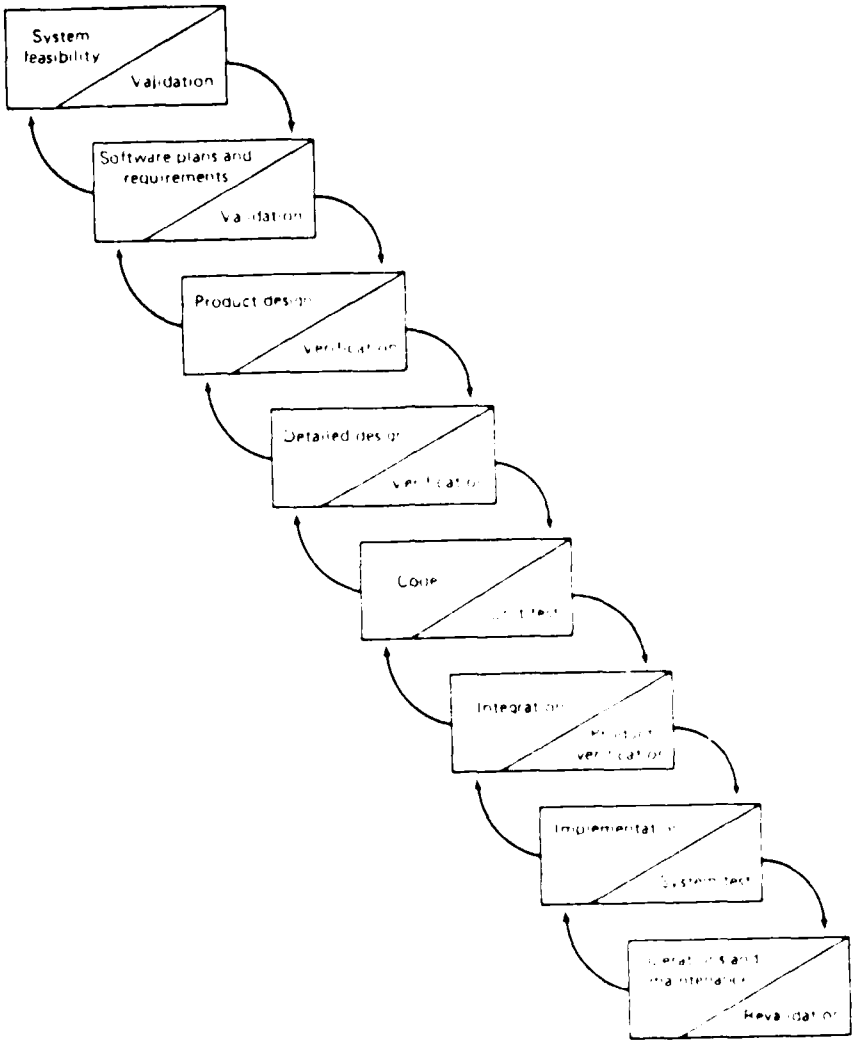


Figure 3.1 Watefall Model of the Software Life-Cycle

6. Retirement phase. The period of time in the software life cycle during which support for a software product is terminated.

The above phases are generally associated with the development of a software system and software maintenance has been religated to those activities performed after development. The simple fact that the cost of maintenance is the dominant factor in the total cost of developing, producing and using software [Ref. 8], warrants a greater examination of this phase than has been expressed in the past. In the early 1960's, it was stated that the IBM personnel working on the development of the Series 360 Operating System did not foresee software maintenance as much of a critical issue [Ref. 5]. They felt that the 360 project would be the ultimate in the computing world and once it was developed, there wasn't going to be much left for them to do. This lack of foresight has dominated the computing industries thinking towards software maintenance for a long time and has encouraged the "second class" citizen approach to all aspects of software maintenance.

All too often managers are initially presented with the life cycle of hardware maintenance and are then asked to apply the same principles to software maintenance. Unfortunately there exists too many critical differences in the two mediums that make any comparison usage invalid. For example, the major objective in hardware maintenance is to return a system to its original state. It is established

that hardware parts have a mean life expectancy and as they wear out they are replaced by a new equivalent piece. On the other hand, the objective of software maintenance is to move a system away from its original state. In the pure sense, software does not wear out. If the environment around a system did not change, then only corrective maintenance would be required to maintain a system forever.

The next alternative for the software manager is to equate the software development life cycle to the maintenance environment. Again, the traditional software life cycle does not fit all the maintenance requirements. The development cycle normally occurs over a time period of one and a half to two and a half years, whereas the maintenance cycle of many changes seldom lasts longer than two or three weeks [Ref. 10]. The development programmer is presented with a long term schedule and milestones to meet and establishes time and programming effort to meet these objectives. The maintenance programmer is often called in the early hours of the morning because a software problem has developed and must immediately react to the critical nature of the system. Software development is the process of building something new, whereas software maintenance is the process of refining something that is already established. Although both processes follow a very similar path, ultimately software maintenance managers must look at maintenance as a separate entity and establish a true maintenance life cycle.

As indicated in Table 1 [Ref. 4], the maintenance process of implementing a change is far more complex and involved than normally considered. The process begins when the need for a change arises and ends after the user has accepted the modified system and all documentation has been satisfactorily updated.

TABLE 1

SOFTWARE MAINTENANCE PROCESS

1. Determination of need for change
2. Submission of change request
3. Requirements analysis
4. Approval/rejection of change request
5. Scheduling of task
6. Design analysis
7. Design review
8. Code changes and debugging
9. Review of proposed code changes
10. Testing
11. Update documentation
12. Standards audit
13. User acceptance
14. Post installation review of changes and their impact on the system

The software maintenance life cycle is a metamorphosis of the traditional "Waterfall" model. It is a pronounced change effected by outside pressure that results in a more compact and more highly crystalline condition. In other words, pressures, such as the time constraints placed on a change or the restrictions of changing someone else's

product, forces the process to be far more compact and defined.

The software maintenance process is repeated many times within the same system. The process of designing, implementing, and evaluating each change occurs over and over again and must be performed in a much shorter time frame than in development. Therefore the maintenance life-cycle of a system is an iterative process, a computational procedure in which replication of a cycle of operations produces results which approximate the desired result more and more closely. It is made up of a series of mini cycles consisting of a design, implementation, and evaluation phases.

IV. SOFTWARE MAINTENANCE PROBLEMS

A. CLASSIFICATION OF SOFTWARE MAINTENANCE PROBLEMS

The realization that software maintenance was the major cost phase of the software life-cycle, has prompted both government and civilian agencies to place far more emphasis in the area of software maintenance. In an attempt to identify the major problem areas in software maintenance, three surveys were conducted; one by the General Accounting Office (GAO) of the U. S., one by the U. S. National Bureau of Standards (NBS) and one by Lientz and Swanson [Ref. 10]. All three of these surveys examined a very broad ADP environment and their ultimate findings are not specifically related to the embedded software environment of the military. However, a number of valuable lessons can be learned and utilized within a tactical support organization. The problems identified in these surveys are grouped into six major areas.

1. Cost

The first problem areas identified is that of cost. The GAO estimated that two-thirds of the programming staff in the U. S. Federal government were involved in some aspect of maintenance. Statistics from the U. S. Department of Defense reported up to 75 percent of the life cycle budget for software systems is related to software maintenance.

Lientz and Swanson [Ref. 11], cautiously estimated software maintenance costs to be over 50 percent of the total software life cycle costs.

Although financial costs are not transparent to the individual maintenance managers in the Canadian military environment, software cost accounting and other related management data could prove invaluable. Unfortunately over the entire thirteen years of the Combined Support Division's existence, little software maintenance cost statistics have been recorded.

2. Management

Management problems have been considered (by the ADP community) to be the second most significant area of concern. Within a tactical software support facility, the problems in this group involve: user and senior level management's perception of maintenance; a lack of goals, standards and performance criteria; forecasting personnel requirements; and managing the user interface. The establishment of set procedures (policy) and the insuring that all maintenance personnel enforce those procedures is an excellent way to avoid problems. Many critical errors have been the result of actions taken for the sake of expediency or from the lack of established software procedures within the Command.

3. Formal Techniques

The third problem area addresses technical approaches and procedures for performing software maintenance. While industry has provided many standards, methodologies, techniques, and procedures for software development, there are few associated with maintenance. A typical response to the reason for maintenance neglect is that "if software is developed well in the first place it will be easy to maintain." This response ignores the fact that sixty to seventy percent of maintenance consists of modifying software to perform new functions or to comply with new requirements [Ref. 8]. It also ignores the sizeable software inventory currently being maintained that was developed without the use of modern programming practices. Without disciplined approaches to maintenance, these systems are characterized by an exponential growth in size and complexity during their operational life [Ref. 10].

4. Software Maintenance Tools

Both the GAO and NBS surveys emphasized the lack of available tools and limited use of tools in supporting maintenance. The CSD would not be an exception to these findings, maintenance tools (which are discussed in more detail in Section VI) have found limited utilization within the Maritime Command. In reality there are few tools available which directly apply and support the unique aspects of software maintenance. In addition, with the majority of

Command's software development being contracted out, tools that are common to development organizations, that would also benefit software maintenance, are rarely transferred to the maintenance organization. It is worthy to note that Lientz and Swanson did not specifically find the lack of software tools as a major software maintenance problem [Ref. 11].

5. Personnel

Personnel issues include such topics as availability, lack of training, turnover, motivation, experience, and scheduling. Throughout the years, the Maritime tactical programming environment has supported the concept of in-house programming by predominately military personnel. Although this has succeeded in providing the operational and user interface needs of the support facility, it has created major software expertise and continuity problems for the programming section. Due to the career requirements of the military system, personnel will normally remain in a programming billet for, at the most, three years. This continual turnover of staff has created a shortage of qualified, experienced programmers and software managers, and the resulting lack of continuity.

In recent years the service has recognized their serious shortage of qualified software personnel and realize that the present military structure is unable to provide for the future growth requirements of the data processing environment [Ref. 1]. To meet the future personnel requirements major policy changes and training restructure are required.

6. Legacy of Development

The last problem area is referred to as the legacy of development. This legacy is that software turned over to the maintainers is often poorly designed, poorly implemented, and poorly documented. Thus, the maintainer has a poorly structured product to maintain without the proper information to understand it. Again, due to the fact that the majority of the Maritime tactical software will be developed by outside contractors, this last area is extremely important to the software support center and desires greater emphasis.

To a varying degree, the fleet software maintenance organization must deal with all six problem areas. It has been difficult to convince senior management that the cost of software maintenance will continue to dominate the overall life-cycle cost and that greater support must be expended in this area. Secondly, personnel shortages are being experienced throughout the computer industry and the service's recent efforts to support the growth of data processing expertise is encouraging. The remaining four areas are the least understood by the military manager and require closer examination within the fleet software maintenance organization.

B. PROBLEMS WITHIN THE FLEET TACTICAL SUPPORT FACILITIES

In addition to the problem areas mentioned in Section A, there exists a number of other critical deficiences that

have greatly effected the efficiency and effectiveness of the Maritime software maintenance organization.

1. Testing Procedures

Testing is a critical component of software maintenance and as such, the test procedures must be consistent and based on sound principles. Secondly, whenever possible, the test procedures and test data should be developed by someone other than the person who performed the actual maintenance on the system. Although each programming team has attempted to test their programs to the best of their ability, there is very few formal testing procedures within the section. The programmer who made the modification usually completes the initial unit testing and then the project manager/editor may complete an informal, free play integration test before it is evaluated in a sea environment. Formal test data is seldom established or retained for future regression test. It is believed that this lack of formal, independent testing has lead to a lower level of user confidence and program reliability.

2. Physical Programming Facilities

An important requirement for the effective maintenance of software is the existence of an adequate working facility. The present facilities for the Maritime tactical programmer are inadequate for any day-to-day computer programming work. The programming section is located on the second floor of an old warehouse building. The space was

divided into a series of double offices with little regards to a programming team organization. The building itself is located next to an industrial junk yard and the daily noise is continual. It was observed that the air conditioning was non-existent and the offices are ridiculously hot and stuffy throughout the entire year. Although there is little research into specifically relating programming environments to productivity [Ref. 12], the physical conditons at the CSD programming section are well below those considered normal for the industry.

3. Software Maintenance Productivity

Faced with a continual shortage of tactical software support personnel and the rising demand for software maintenance services, the requirement to improve programmer productivity has taken on a higher priority. During an initial visit to the Maritime software maintenance facilities at CFB Halifax, Aug 85, an attempt was made to establish whether productivity was a major problem within the tactical programming environment, it was observed that the present level of productivity was an extremely difficult factor to estimate. Within the CCS 280 system, lines-of-code (LOC) "added or changed" could be counted per revision and they could be separated by comment and non-comment lires, but LOC "deleted" was only recorded as a combination of both. As for the effort expended on each revision, personnel resources and their individual times on the system were non-existent.

With no formal recording of programming effort to specific maintenance task and no established productivity measurement, an accurate evaluation of productivity became impossible.

V. SOFTWARE MAINTENANCE MANAGEMENT: GOALS AND ISSUES

A. PERFORMANCE GOALS OF MARITIME SOFTWARE MAINTENANCE

The overall goal of the Fleet Tactical Support Center, maintenance organization can be summarized as "keeping all Naval Shipboard Software Systems up and running according to their specifications, and responding to the changing computer requirements of the fleet."

In order to evaluate the success of this maintenance organization in meeting these general goals, a few key elements must be measured [Ref. 10]:

* Systems availability and reliability,
* User satisfaction,
* Change request/problem report response time,
* Productivity improvement.

In the past these measures have played key issues in establishing priorities for the maintenance effort, however there has been no formal evaluation within any of these areas.

1. Increasing System Availability and Reliability

The U. S. National Bureau of Standards has stated that "Problem report frequency is a good measure of how well a maintenance organization is doing." [Ref. 10] Within the Maritime tactical programming environment, this statement would have proven incorrect. The sailor at sea has approached

the computerized Command and Control systems as a "black box." With the Navy's "can do" attitude towards any task, they have continually worked around many faults in the programs and never report the errors. Once alongside, ship's staff would report that they had to reload a program every so many hours due to its degeneration, but specific problem examples were extremely hard to establish. The amount of downtime that results from a failure is an indicator of the severity of the problem, but it is hoped that these critical errors would be resolved prior to any operational release. A better measurement of the system's reliability has been to accurately record the operational up-time between the requirement to reload a system due to degradation. If the frequency of reload requirements is steadily decreasing, the maintenance organization is performing maintenance more effectively.

2. Encouraging User Satisfaction

Although difficult to evaluate objectively, user (fleet) satisfaction is the most important performance measure of a maintenance organization. The overall value that the software product provides the end-user is the only true measurement of its effectiveness. In addition, the user's response should direct the maintenance organization in setting priorities as to those areas requiring greater software maintenance effort. In dealing with the shipboard personnel, the user will lack in-depth knowledge of data processing,

but the operators current operational expertise and cooperation are invaluable in the production of a viable system. It is imperative that the support facility maintains an effective user interface.

In the past, the Command scheduled bi-annual meetings between the senior ship personnel and the programming staff. At these open forum meetings, called the Maritime Operational Shipboard Software Working Group (MOSSCOW), the maintenance staff presented the status of program change requests and the ship's personnel would be able to present their approval for the proposed changes. The meetings formalized the exchange of ideas and information between all parties and proved very effective for the small number of systems involved.

As the number of shipboard systems continue to increase, the need for more individual user/maintainer interfaces will have to be explored. A technique that was observed to be very effective in assessing the fleets satisfaction was to have the programmer periodically go to sea and observe the software product in its operational environment. The feedback that the programming staff obtained, by sitting alongside the ship's "middle watch" operators, proved far more valuable than the formalized reporting procedure.

3. Improving Change Requests/Problem Report Response Time

The time it takes to make a change to the system significantly influences the attitude about the maintenance organization. In a military, peace time environment, only

corrective changes, that are critical to the running of the system, take on an urgent schedule. All other changes are performed based on the resources available to the system. Users want a response to their requests as soon as possible and accurate measurements of response time can assist a manager in assigning programmer resources.

The change request and problem report forms can be used to capture statistics about response time. These statistics can be used to justify additional tools to increase productivity and to identify bottlenecks in administrative procedures.

4. Quantifying Productivity Improvement

Due to the lack of historical data, the author's initial attempt to establish a productivity factor for the CCS 280 programming section proved impossible. Although a great deal of changes have been made over the past thirteen years, statistics on the amount of changes made and the effort expended in making them, are non-existent. Productivity measures have received little emphasis within the Maritime programming organization. Although there is no universally established standard or commonly accepted productivity measure, using some measure of productivity will aid in pinpointing areas where productivity gains can be realized.

B. CRITICAL ISSUES IN MANAGING SOFTWARE MAINTENANCE

A tactical software maintenance manager must be both a good technician and a good manager. The effective use of

good management techniques and methodologies in dealing with scheduling maintenance, interfacing with the end-users, coordinating the maintenance staff, and instituting the use of the proper tools and disciplines is essential to a successful software maintenance effort. Within the service, military managers have usually received the managerial skills required to coordinate people, however the software maintenance manager has also got to be able to control the technical aspects of the environment. Without the technical background and "hands on" experience in performing software maintenance, the manaer will not be able to appreciate the conflicting needs and requirements involved in maintenance tasks.

Control of software maintenance requires understanding of the software system involved, the users of the system, and how the system interacts with the user's environment. Also, consideration must be given to the talents of the maintenance team members, and to control of maintenance through configuration management, change control, software quality assurance and communications among both the team and between the team and the user organization.

Configuration management, change control, software quality assurance procedures and communications, when used in unison, form a stable environment in which maintenance can be performed in an efficient manner. By implementing these basic maintenance control procedures, software

maintenance can be managed with a high degree of visibility and control.

1. Configuration Control

Configuration control is an essential element in managing tactical project which contains numerous change requests. By grouping changes into a number of distinct software releases, it is possible to provide users with a relatively stable environment in which the ongoing changes are well understood by all persons involved, and new errors or unexpected features are minimized.

Controlling the activities performed by the maintenance personnel is best accomplished when there is a definitive set of procedures to follow and enforce. Within Maritime Command regulations, problem reporting procedures are already well established. Also the reporting forms, utilized by the fleet, have proven to be effective and have provided adequate data for all administrative requirements. Within this area the only shortcoming is the lack of analysis of these problem reports. Data gathered by the problem reporting system could be very useful in describing the changing quality of a system. The project managers should examine each module in a system for a frequency distribution of the number of problems by module. Since, experience has indicated that sources of errors are rarely uniformly distributed across all modules [Ref. 13]. In fact, within a system a subset of modules will exhibit high error rates. Once these

problem areas have been established, management is better able to determine whether it would be more cost effective to redesign rather than continue maintaining poorly designed/ programmed modules.

2. Change Control

Change control can be made easier by implementing software tools which control access to source code. By using a program library, and checkout/checkin/install procedures [Ref. 14], it is possible to track changes to source code and prevent simultaneous modification of the same file by more than one person.

3. Software Quality Assurance

Another control technique that requires attention within the fleet programming section is Software Quality Assurance. Quality assurance is a planned and systematic pattern of all actions necessary to provide adequate confidence that the product conforms to established technical requirements [Ref. 3]. A Software Quality Assurance (SQA) plan should document procedures to identify the documents that will be reviewed, the personnel involved and the schedule required to meet all product requirements prior to release. The SQA procedures should, as a minimum, involve design and code walkthroughs, and they may also include a documentation audit, test monitoring, and an SQA sign off. Although each programming team does utilize code walkthroughs, there is no formalized SQA procedure in the section.

4. Communications

A critical part of effective maintenance control is communications, both among maintenance personnel and between the maintenance and fleet organization. Without providing good communications channels, it will be difficult to apply configuration management and change control methods successfully. Consistent and accurate use of the software problem report (SPR) and the program change requests (PCR) is an excellent way of following changes from initial request to final resolution. They provide a structure within which change control boards and management teams can fully discuss changes during progress reviews.

The primary objective of software configuration management (SCM) is the release of operationally correct, reliable, and cost effective software [Ref. 10]. SCM is responsible for configuration identification, configuration control, status accounting, auditing, records retention, disaster recovery, library activities, and coordinating the various activities with the users, management, and the staff. Within a programming team, these duties are split between the project manager and the editor, with each team establishing their own procedures. A single SQA plan and SCM procedure for the entire section is required to standardize the overall support policy and to ensure a high quality of software products being released to the fleet.

VI. SOFTWARE MAINTENANCE TOOLS

A. CLASSIFICATION OF SOFTWARE MAINTENANCE TOOLS

Software tools are computer programs which can be used in the development, analysis, testing, maintenance, and management of other computer programs and their documentation [Ref. 4].

As detailed in Table 2 [Ref. 4], software tools cover a wide range of products that can be utilized to a varying degree in a software maintenance environment. Software tools can be divided into technical or management categories. Technically, they can assist in processing, analyzing, and testing systems as well as those which help the maintainer manipulate and change the source code and the documentation. The management tools assist the maintenance manager in controlling and tracking all of the maintenance tasks. In a brief "interview" survey of the tactical programming section, it was observed that many project managers were not aware of the availability of the variety of tools for maintenance activities and most expressed a desire and need to explore the user of more software support tools. This section will attempt to identify the major types of tools and how they can be utilized to assist the Fleet Tactical Software Center.

TABLE 2

SOFTWARE MAINTENANCE TOOLS

	TOOL TYPES	FUNCTIONS
1.	Support Librarians	formatting, configuration management
2.	Code Analysis	cross-reference, data flow analysis, structure checking, comparison, interfaces analysis, type analysis
3.	Requirements Data Base and Tracing	completeness checking, consistency checking, traceability
4.	Test Data Management	test data generation, regression testing
5.	Test and Error Analysis	problem report data base
6.	Code Instrumentation Execution Analysis	coverage analysis, tracing, symbolic execution, debugging support
7.	Text Editors	editing
8.	Structure/Restructure	translation, restructuring
9.	Application Generators	synthesis
10.	Optimizers	optimization
11.	Simulations/Emulations	simulation
12.	Management Tools	schedulers, PERT/CPM, status reports cost estimation models

1. Support Librarians

There is a wide variety of tools which control changes to the configuration of the source code. They range from specific configuration control systems to programmer utilities which facilitate standard program construction and modification processes. The Maritime maintenance organization does practice configuration control, but only on a manual basis, with each programming team doing their own standard of configuration management. This is an area in which automated support librarian tools would be extremely useful.

2. Code Analysis

The lack of up-to-date system documentation has forced the maintainers to rely heavily on the source code. When debugging any reported error, the programmer has had to first go to the source code to evaluate the situation. Although this has been proven not to be the most productive method of debugging [Ref. 13], the source code has become the only reliable documentation within the section. Under these circumstances, code analysis tools provide an even greater benefit. Code analysis tools can be used to process the source code for: standards enforcement, data flow analysis, complexity analysis, generation of cross reference listings, or production of flow charts.

3. Requirements Database and Tracing

Faced with the requirement to continue maintaining two older systems that were developed under an unstructured

programming environment (CCS 280 & UCS 280/257), many changes to a function or capability of the system can not be easily identified with a particular routine. Tracing a change through the design of the system to where it is implemented can be time consuming. Tools exist that provide this type of traceability. They can be very advantageous to the maintainer attempting to analyze how to modify a system and what impact a change will have.

4. Test Data Management

As previously outlined, the lack of testing procedures and test data generation/retention are major problem areas within the Maritime maintenance organization. Test data may be prepared manually, generated from actual production data sets, or prepared as a result of analysis of the code. During acceptance testing a great deal of test data is produced by the contractor and should be made available to the support organization for regression testing. Tools such as test data generators and data base management systems support maintenance testing activities and provide considerable cost savings by preventing the maintainers from generating and storing test data manually.

5. Test and Error Analysis

If utilized correctly, problem reports from the end-users are valuable sources of information to a maintenance organization. They not only identify corrections that must be made but help keep track of the frequency, number, and

type of problems being reported. Although the Command has an adequate procedure for the fleet to report errors, it is better to encourage the ship's personnel to complete the reporting forms when an error has occurred. Far too often an operator will just work around the problem and later forget to report the occurrence. End-users should be made aware that by reporting all suspected errors the maintenance organization is able to better improve the program's performance over time. This improvement can be realized by completely redesigning and rewriting programs which exhibit high error rates, by assigning senior personnel to those problems, and by instituting standards which will alleviate occurrences of certain types of errors during modifications or correction activities. As well as the end-users participation, the maintenance organization must analyze all errors and retain the results of same in order to improve their products performance.

6. Code Instrumentation/Execution Analyzers

Tools exist which instrument source code with counters, sensors, and assertions, and then provide reports after test cases have been run against the code. These tools report code and path coverage and assertions during the test/execution run providing some assessment of the thoroughness of testing. Other tools in the category assist in optimization by identifying frequency of execution of individual statements and segments of the code. Such tools support the

debugging process by allowing a programmer to step through the execution of a program during execution. These types of tools provide automated support in analyzing the dynamic behavior of the code, and are particularly helpful when tuning a system or during testing (regression testing) changes to the system. It is impossible to find all the errors in a program prior to its release to the fleet, but these tools can help ensure that all segments within the program have been considered. There utilization can greatly improve the overall confidence in the program and its reliability.

7. Text Editors

Text editors are already used in the [illegible] ware environment to produce documents and to enter and modify code. An additional benefit could be realized in the support center if the documentation produced during development could be delivered in machine readable form. This would enable the modification of the documentation to become a process similar to that of code modification and a far easier process to enforce upon the programming staff. Any tool that can assist in keeping documentation up-to-date can not help but provide a significant savings in debugging and designing.

8. Structuring/Restructuring

In recent years a number of products have been advertised to generate structured code from unstructured code. The benefits of working with a structured product are well

understood by all programmers and can be valuable in many aspects of a maintenance organization [Ref. 4]. However, the capabilities of these structuring tools are extremely new and unproven. It is considered that, in most cases, the inventory of unstructured code would have to be considerable to warrant the expense of a restructuring tool. If all new program development demanded the use of structured programming techniques, these tools would be of low priority to the support center. Other tools in this category also include structured language preprocessors to facilitate structured coding and language conversion packages which support conversion of code.

9. Application Generation

Application generators are high level languages oriented towards a specific type of application which allows use of functional statements to build a system. This tool's use is dictated to the maintainer if the development was performed using one. Since most of the Maritime tactical software development is performed by outside contact, the enforced use of a standard application generator would prove difficult.

10. Simulation/Emulation

Simulation tools are extremely familiar to the Maritime tactical support center. Duplicate sea/shore facilities with their simulated inputs/outputs are the primary test beds for assessing the performance of all supported systems.

11. Management Tools

There are numerous tools which can assist in managing a maintenance organization. Tools such as schedulers can be used to keep track of different tasks, personnel assignments, milestones, etc. Tools that utilize data from various sources can provide management data which describe the amount of change being made to a system, the growth in complexity to that system as a result of the changes, the performance of the maintenance organization in terms of user problem reports, and response time for completion of change requests. Although these tools may not be directly related to an increase in maintenance productivity, they do assist the manager in making better decisions about resource allocation.

B. INTEGRATING SOFTWARE MAINTENANCE TOOLS INTO THE MARITIME ENVIRONMENT

With the wide variety of software tools available on the market, the question of which ones to employ in a software maintenance environment is difficult. As noted in the problem section, tools that are common to develop the software are often very beneficial to the software maintenance organization. Development contacts should be designed to enable the transfer of useful software tools to the fleet tactical organization. In addition, if a military personnel could be employed as a member of the development team and upon the system delivery, he is posted into the maintenance organization, the transfer of these tools and system knowledge could be far easier.

Another approach to realizing long term benefits from software tools is to accumulate an inventory of tools that will support the various activities performed within the service. The goal should be to establish an integrated set of tools which support a logical sequence of growth activities. In order to maximize the phasing in of software tools due to budgetary/resource constraints, Table 3 provides a priorized list of tools which should be accumulated.

TABLE 3

SOFTWARE MAINTENANCE TOOL PRIORITIES

First Phase	Structured Language Compilers (e.g., ADA) Test Editor (already available) Configuration Manager/Support Library On-Line Debugger
Second Phase	Test and Error Analysis Management Tools
Third Phase	Code Analyzer Optimizers Code Instrumentation/Execution Analysis Simulation/Emulation
Fourth Phase	Restructurer Requirements Data Base Application Generator

VII. THREE ACTIVITIES FOR IMPROVING THE FLEET TACTICAL SOFTWARE MAINTENANCE

A. MAINTENANCE DOCUMENTATION

Under the problem area entitled "Legacy of Development," poor documentation has had the most lingering effect. Poor design areas and poor implementation are quickly pinpointed by the fleet personnel and usually corrected under the initial contract, but incomplete/inaccurate documentation is harder to detect and the eventual correction falls to the maintenance organization. At the November 85 IEEE Conference on Software Maintenance [Ref. 5] it was stated that:

> Software Documentation was something that developers will not do (correctly) and something that maintainer will not read.

A brief examination of the four major Maritime shipboard software programs quickly demonstrated support for this statement. From the initial CCS 280 project to the newly introduced ADLIPS system, the contracted developers ensured completed documentation, but, as the schedule and costs increased, the accuracy and value of the software documentation suffered. Within the CCS 280 system, the inability to easily change the documentation, forced the system and user manuals to grow out-of-date, making the source listings the two only reliable documentation for the system. Even though the ADLIPS developers were aware of the need for accurate,

complete documentation, the numerous "last minute" changes, the lack of qualified evaluation personnel acceptance periods, made the value of the eventual system docmentation questionable.

Ideally documentation of a system should start with the original requirements and design specifications and continue throughout the life cycle of the system. Good software documentation is essential to good maintenance. The extent of documentation to be maintained is a function of management practices and the size, complexity and risk of the project [Ref. 15]. Due to the Maritime Command's contractual methodology in developing its tactical software, the formality, extent and level of detail of documentation has varied greatly with each system. In the past, the level of documentation has been at its highest when it was initially accepted from the developer and steadily declined with each change. The software maintenance manager has had to evaluate each project's documentation as it was delivered and has had to bring it up-to-date or maintain it at an acceptable standard established within the Command.

Due to the limited manpower resources within the service it is important for the developing agency, (HDHQ), to establish documentation standards and criteria for the service. Within these guidelines, the overall value of the documentation must be assessed and the question as to who is to benefit from its upkeep. Documentation that appears useless to

the maintainer will not be maintained accurately or up-to-date.

The success of a software maintenance effort is dependent on how well information about the system is communicated to the maintainer. Documentation should support the useable transfer of pertinent information. Documentation guidelines should include instructions on what information must be provided, how it should be structured, and where the information should be kept. In establishing these guidelines and standards, one must remember that the purpose is to communicate necessary, critical information, not to communicate all information. Important documentation guidelines are [Ref. 4]:

1. Keep it simple and concise;
2. The maintainer's first source of documentation is the source code. Thus, if priorities have to be established, this area should be of the highest quality of documentation;
3. The manager's first source of docuemntation is the design specifications and implementation reports;
4. The user's first source of documentation is the Users Guide and the maintainer;
5. Documentation maintenance is a vital part of system maintenance; and
6. Documentation cannot be "almost correct." Either it is up-to-date, or it is useless.

A second invaluable factor in the success of any documentation is that not only must the necessary information be recorded, but it must be easily and quickly retrievable by the maintainer. On-line documentation which has

controlled access and update capabilities is presently the best form of documentation for fulfilling this requirement.

Today there exists a number of methodologies which stress differing formats and styles. Each methodology has its strength and weaknesses and for individual maintainers the preference may differ on which methodology is best for his requirements. The important element for the Fleet Support Center is to adopt a documentation standard and to then consistently enforce adherence to it for all software projects.

B. TESTING PLANS AND PROCEDURES

1. Validation, Verification, and Testing Plans

The proliferation of software and the increasing complexity of embedded software systems has increased the need for software maintenance and the importance of software maintainability. An integrated validation, verification, and testing (VV&T) program for software is essential to achieving improved software maintenance and maintainability. [Ref. 16]

When discussing the maintainability of a program it is impossible to separate the development and maintenance life-cycle phases of a program. As stated in FIPS 101 [Ref. 17], "a software system should be developed with maintenance in mind and maintained with future maintenance in mind." A well designed VV&T policy throughout the entire software life-cycle can greatly contribute to the overall goal of maintainability of the software system.

FIPS 101 presents a VV&T methodology recommended for use throughout the software life-cycle to ensure the production and maintenance of quality software. This methodology is an organized collection of procedures to review, analyze and test the software. The following definitions are extracted from FIPS 101 [Ref. 17]:

* Validation determines the correctness of a product with respect to the software requirements.

* Verification employs integrity and evoluation checking to determine the correctness of the products at each phase of development or maintenance with a previous phase. Integrity checking requires analysis of the internal consistency and correctness of the products. Evolution checking ensures the completeness and consistency of the products at different development phases, where one product is a refinement or elaboration of the other.

* Testing, either automated or manual, examines program behavior by executing the program and sample data sets.

The analytic techniques of a VV&T program generally fall into three categories: static, dynamic, and formal. Static analysis detects errors through the examination of the product but not its functional or computational aspects. It is also the technique used to examine documentation items at all pahses of development and maintenance. Examples are code reading, data flow analysis, auditing, complexity measurement, consistency checking, review, and inspection.

Dynamic analysis is the process of determining the correctness of a product and detecting errors through the study of the response of the program to a set of input data. It addresses the functional, structural and computational

aspects of the program. Examples are assertion checking, constraint evaluation, coverage analysis, simulation, timing, and tracing.

Formal analysis is the use of a mathematical discipline to analyze the algorithms or properties of the software. Examples are predicate calculus and symbolic execution.

The VV&T analyst may use these techniques on software code, software documentation, or any other software product. It is important to remember that no single VV&T technique can guarantee correct, error free software but, when carefully applied, a combination of techniques can provide confidence in the adequacy of the software.

The establishment of a complete VV&T plan will require the cooperation of agencies outside the Fleet Tactical Support Center. Without their support during the development phases, many of the benefits of such a plan will be limited. The benefits to software maintainability that the FTSC can receive from a VV&T plan are listed below and described in more detail in Reference 18:

* VV&T plan provides the opportunity to ensure that pro ducts necessary for maintenance VV&T will be deliverables;
* VV&T techniques and tools in place;
* Standards and guidelines in place;
* Testability of existing software;
* Clearly defined, well documented software;
* Test cases for regression testing;

* Test cases, to test modifications;
* Assurance in quality of the software units;
* Metrics to aid reuse or rewrite decisions;
* Some VV&T independence from maintainer;
* Enhanced maintainability.

When a software project begins with a sound VV&T program, it is easier to carry a VV&T program into future maintenance of the software and contributes continuously to the maintainability of the software.

2. Test Procedures

Within the tactical programming environment, testing procedures, like documentation, have been extremely unstructured. All too often individual programmers have initiated, designed, implemented and tested a change, completely independent of any outside monitoring or influence. Formal testing procedures have been non-existent and project leaders have failed to appreciate the true value and purpose of an overall testing policy.

Glenford J. Myers defined testing as "the process of executing a program with the intent of finding errors." [Ref. 10] Too often software personnel have misunderstood this definition and have errors are not present or to show that a program performs its intended functions correctly. These misconceptions have allowed programmers to limit their scope of testing which has lead to incomplete testing procedures and error-prone operational programs. For example,

programs that do what they are supposed to do can still contain errors and a process of demonstrating that errors are not present is impossible to achieve for virtually all programs. Myers emphasized that a successful test case is one that finds an error, and that an unsuccessful test case is one that causes a program to produce the correct results.

The following is a list of testing principles that Myers considered imperative to all software testing environments [Ref. 13]:

* A programmers should avoid attempting to test his own program;
* A programming organization should not test its own programs;
* A necessary part of a test case is a definition of the expected output or result;
* Test cases must be written for invalid and unexpected, as well as valid and expected, input conditions;
* Examining a program to see if it does not do what it is supposed to do is only half of the battle. The other half is seeing whether the program does what it is not supposed to do;
* Avoid throw-away test cases unless the program is truly a throw-away program;
* Do not plan a testing effort under the tacit assumption that no errors will be found;
* The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section; and
* Testing is an extremely creative and intellectually challenging task.

The majority of these principles are not being followed within the present Maritime tactical programming

environment. An independent VV&T organization must be established to remove the unavoidable testing biases of the original programmer. There should be a standard approach for testing and using automated aids for test case generation and storage. There should be a commitment to establish and maintain a standard set of test case (data) for each application. This set of test data can be used for regression testing whenever software modifications are required. By simple adopting these princples, the Maritime Command can create a more effective testing environment to ensure the software quality throughout the maintenance life-cycle.

C. SOFTWARE MAINTENANCE PRODUCTIVITY

1. Productivity Measurements

In order to improve the software maintenance productivity in the Fleet Tactical programming environment we must first be able to define and measure the present productivity of the software organization. In the Maritime software maintenance environment, this has proven to be an impossible task. Firstly there does not exist an established, acceptable unit of measurement for a programmer's productivity and secondly the required production data has not been recorded on any of the various projects within the service.

In recent years the IEEE society has attempted to establish a standard productivity metric for the computer industry. Although the standard has not yet been officially published, they have proposed the following definitions:

Software Productivity:	The ratio of units of software to the cost of its production.
Software Productivity: Metric	A quantitative measure of software and its cost of production. There are three types: * Those which define the units of cost, * Those which define the units of software, and * Those which define the ratio.

The ability of quantify software productivity could prove very valuable to the tactical software maintenance organization. By establishing quantifiable productivity metrics, system managers could evaluate individual programmer performances or their overall system release productivity. They could indicate productivity trends within their project and allow comparisons with other projects. Also productivity metrics could be used to predict future change requirements. Knowing the capability of your maintenace team will produce far more accurate estimates and resource allocations.

The most common form for a productivity metric is:

VOLUME/EFFORT

In a software maintenance environment the most obvious volume metric would be to count the number of lines of code added/changed/deleted from the program. However lines of code (LOC) has proven to be a very questionable factor. For example, is every comment line in a program counted, or do we only count executable code? What do we count if a

programmer is inserting "reusable" code and can we count code that has been produced by a program generator? Do we apply a language conversion ratio between specified programming language?

The volume metric could have other variations. A complexity factor could be substituted into the formula. Halstead, McCabe, Albrecht, Henry and Kafura, and Basili and Hutchens, all developed methods to quantify a program's complexity that could be adapted to a productivity metric. The effort metric is always the manpower resource measured by person months or person hours.

Another productivity measurement technique is to normalize the project data to a mathematical model, and to use the coefficients of the model as the productivity metric. Examples of cost modeling systems, which are designed to forecast software resource and effort requirements, are [Ref. 9]:

* Norden - Rayleigh curve
* Putnam - SLIM
* Parr - Sech squared curve
* Boehm - COCOMO
* RCA - Price-S

Due to the wide variance in this area and the fact that the Maritime Command has little experience in productivity metrics, it probably does not really make too much difference which productivity measurement the software

maintenance organization chooses. Initially choose one or two reasonable measures and experiment with the collection of data and their results until an evaluation of the methods can be made. As stated by Norman Lyons [Ref. 12]:

> Whatever measure is chosen, it should be applied with common sense, examined frequently and compared with other measures of productivity. Slavish adherence to an inappropriate productivity measure could do more damage to real productivity than not paying any attention to productivity at all.

2. Programmer Productivity

The lack of historical software maintenance measurements throughout the data processing industry has forced most maintenance managers to examine software development data in order to evaluate their specific research requirements. A. J. Thadhani conducted research into programmer productivity during several IBM application development projects [Ref. 19]. A number of useful productivity factors were presented that could be applied to a military software maintenance organization;

a. Interactive users (programmers) were twice as productive when the computer response time for human-intensive interaction was 0.25 second instead of 2.0 seconds. For all the engineers, irrespective of their level of expertise, it took more than twice as long to complete the task at the higher response time. This phenomenon seemed to be related to an individual's attention span. The traditional model of a person thinking after each system response appears to be inaccurate. Instead, people seem to have a sequence of actions in mind, contained in a short-term mental memory buffer. Increases in system response time seem to disrupt the thought process, and this may result in having to rethink the sequence of actions to be continued.

b. Programmers spent a significant part of their workday at their termials, between four and five hours, with no evidence of fatigue. Moreover, their interactive work was dominated by human-intensive work like file edit and browse. During the design, coding and testing phases, 90 to 95 percent of programmer terminal time was human-intensive, with productivity dependent on short response times.

c. The skill and experience of the development team and the complexity of the program are significant factors affecting programmer and project productivity. Experienced programmers were two to four times more productive than the less-experienced. In addition, in-expereienced programmers consumed one-third as many computer cycles to generate an equivalent number of lines of code.

d. The data suggested that the time spent by a programmer in developing a module may indicate a level of difficulty experienced by the programmer relative to his other modules and may be a useful indicator of perceived complexity.

As discussed earlier, the life-cycle of software maintenance consists of a series of short design, code and test phases. Thadhani's studies of application development concentrated on these three phases, which make the findings more closely aligned to software maintenance. The Maritime Command organization may also benefit from this study. Fast response time is just one component of the need for good computer services. Good computer services, such as 24-hour continuous availability, response times of less than one minute to small foreground complies and executions, and turnaround times of less than 15 minutes for small batch executions, can all contribute to more productive programmer and project productivity.

If a programmer is spending over 90 percent of terminal time, in the human-intensive functions of edit and

browse, more effort should be expended at making these processes as easy and convenient as possible. Military programmers must have a high ratio of terminal to programmer availability and easy access to the computer resources.

Current embedded software maintenance processes are labor-intensive and require highly skilled programming expertise. Experienced programmers have displayed a significantly greater productivity factor [Ref. 19] and, with the service facing an ever increasing demand for computer personnel resources, an objective of training and maintaining a high degree of programmer expertise should be paramount.

Lastly the apparent relationship between the time a programmer spends on a module and its complexity may prove useful in the maintenance environment. For example a useful testing strategy would be to extensively test those modules on which the programmer has spent significantly larger amounts of time. Here the complexity of the program may prove the greatest and most prone to error.

VII. SUMMARY AND CONCLUSIONS

Although the Maritime Command entered the embedded software environment in the early 1970's, they have failed to grow and develop with the modern advancements in software maintenance technology. Their recent decision to reorganize the entire Naval Shipboard Software process and establish a new Fleet Tactical Software Center has demonstrated their awareness of the growing software maintenance problems and their desire to implement improvements in these areas.

The software maintenance life cycle is a metamorphosis of the traditional "waterfall" model, which is far more compact and defined than the development process. It is an iterative process of design, implementation and evaluation which approximates the desired results more and more closely. Therefore many of the principles of development can be incorporated into a successful software maintenance environment, but they must be closely examined and refined to meet the unique demands and requirements of the maintenance organization.

A review of the most prevalent problems within the overall software maintenance industry revealed that the tactical programming division is not an unique software maintenance environment in that it experiences similar difficulties as those of its civilian counterpart. While software maintenance

problems were both managerial and technical in nature, managerial factors and capabilities were the most important factors in the improvement of the software maintenance process. Although the majority of software tools and techniques were designed for the development phases, management must take a closer look at how software is maintained and employ those tools and techniques that can be effective in the software maintenance organization. The lack of trained, and experienced personnel is a problem throughout the entire computer industry. The service's recognition of the continual turnover of personnel, and its accompanying lack of continuity within the systems, has proven to be encouraging to the resolution of this situation. Lastly, the problems associated with the "Legacy of Development" are the least controllable without the full cooperation and assistance of the acquisition and development agencies. Qualified software personnel with direct monitoring and control of the contracted product are required to resolve these poor design, implementation and documentation problems.

Two problem areas, that were specifically identified, were those of software testing and productivity. Within the tactical environment, the formalized testing of software has been a very ineffective operation. Several of the basic principles of software testing have not been adhered to and the resulting product's quality and reliability have continually been below the level required. Secondly, the

question of productivity within the service could not be truly addressed due to the complete lack of data collection and, productivity standards and procedures.

Prior to the Fleet Tactical Support Center implementing improvement policies and procedures it must be able to evaluate its present performance and ability to meet its goals. The FTSC must establish a formalized procedure for data collection in order to evaluate the systems availability, reliability; user satisfaction; change request/problem report response time; and productivity. With minimal resource requirements, this historical data can greatly assist a manager's decision making process.

In order to maintain control over the software maintenance process and to ensure that the maintainability of the system does not deteriorate, it is important that a firm discipline is placed on how software maintenance is performed, and that discipline is enforced. Software maintenance must be performed in a structured, controlled environment. It is simple not enough to get a system "up and running" after it breaks. Proper management control must be exercised over the entire process. For the FTSC, standard control procedures must be incorporated for the entire section verse the individual systems meeting their own needs.

Although the majority of the tools and techniques available to the software maintenance environment were designed for the development phases, many of them can be applied to

maintenance requirements. The problem facing the maintenance manager is which methods will provide the greatest benefit and how can they be implemented into his organization. A wide variety of techniques and tools have been present and their basic advantages have been outlined in Sections VI. A recommended approach to realizing long term benefits from software tools is to accumulate an inventory of tools that will support the various activities performed by the maintainers. By establishing an integrated set of tools which support a logical sequence of activities, share data, and provide a common user interface, the FTSC can obtain the maximum benefit for all systems. For example automatic support library/configuration management tools are required now to make standardizing change/configuration control far easier for the entire section. As the set of tools grows, code analyzers, test and error analysis and other management tools can be added to improve the quality and reliability of the software product.

Within the Maritime environment a large percentage of the embedded software systems being maintained, were not designed to accommodate change. In the past the software support center has had little influence on the quality of the product being delivered from the development process. The maintenance organization has then had to expend a great deal of energy trying to improve a program's maintainability, but unless maintainability is engineered into the original

software product, their efforts have been far less productive. Maintainability must be established during the requirements, design and development stages and built into the software using standards, and design and development techniques. The transition to the maintenance phase can then be made easier by incorporating many of these development tools and techniques, along with the use of more effective maintenance methods.

The upkeep of documentation is a major problem within the Maritime tactical programming environment. With the numerous different tactical systems being delivered from various contractors, uniformity in the standard of documentation for the Maritime Command to maintain adequate documentation for all its system, it must initially establish what standard of documentation must be maintained, remembering who it is to benefit and the man-power cost that will be required to maintain it at that level. The second requirement to make it easily and quickly retrievable to the maintainer can be resolved by adopting on-line documentation techniques within the support organization.

Due to poorly defined Verification, Validation and Testing procedures with the Maritime Command, the CCS 280 programs took over ten years to become a reliable command and control system. The lack of an overall VV&T plan for the entire life-cycle of a system has made the software maintenance task extremely difficult. In order to establish a

viable VV&T plan, a great deal of cooperation and mutual support will be required between the acquisition organization in NDHQ and the software support facility on the coast. Once a system is accepted into its operational environment, the support center must recognize the value and requirement for an independent testing group. Formalized testing standards and procedures for all program releases will benefit both the effectiveness of the maintenance organization and the reliability of the fleet product.

When employed correctly, productivity measurements can be very useful to the efficient operation of a software support facility. Although a standardized productivity metrics has not been established for the computer industry, a number of useful systems are readily available and easily implemented. These systems can supply a manager with the tools to better evaluate a systems present status and effectively estimate and allocate the project's valuable maintenance resources. In order for an organization to improve its productivity, it must first understand the factors which effect the programmer's performance. Good computer services, which include both the physical facilities, as well as the system availability, reliability, accessibility, response time, and batch and print turnaround times, are all factors that effect programmer productivity. In addition, knowing how a programmer utilizes his working hours can assist the manager in the better utilization of his production effort. The

skills and experience of the programming personnel is very important to the capability of the maintenance organization. The importance of this requirement on productivity must be addressed by high level Command and the posting and promotional policies of the service must be changed to encourage longevity within the tactical programming environment.

This thesis was designed to assist a new software maintenance manager in better understanding the important aspects of tactical software maintenance management. Knowing what occurred in the past and the mistakes that are prevalent throughout the industry can help guide him in his dealing with software maintenance. The industry can only be considered in its early development phase and changes can and are happening daily. The establishment of the Fleet Tactical Support Center is an important step in the growth of the software support capability. It is now up to the maintenance managers to implement the steps to improving the maintainability, reliability and capability of the Maritime Operational Shipboard Software.

LIST OF REFERENCES

1. Naval Shipboard Software - The Way Ahead, DGMEM National Defence Headquarters, Ottawa, Ontaria, K1A ØK2, 1984.

2. Establishment of Fleet Tactical Software Centre (FTSC), (DCOS P & T) Maritime Command Command Headquarters, Halifax, N. S., B3K 2XØ, 1984.

3. "IEEE Standard Glossary of Software Engineering Terminology," Institute of Electrical and Electronic Engineers, Inc., New York, NY 1983.

4. Martin, Roger J., and Osborne, Wilma M., "Guidance on Software Maintenance," NBS Special Publication 500-106, National Bureau of Standards, Washington, DC, 1983.

5. Proceedings - Conference on Software Maintenance - 1985, IEEE Computer Society Press, Washington, DC, 1985.

6. Swanson, E. B., "The Dimension of Software Maintenance," Proceedings of the 2nd International Conference of Software Engineering, IEEE Computer Society Press, Washington, DC, 1976.

7. Osborne, Wilma M., "Executive Guide to Software Maintenance," NBS Special Publication 500-130, National Bureau of Standards, Gaithersburg, MD, 1985.

8. Proceedings - Software Maintenance Workshop - 1983, IEEE Computer Society Press, Silver Spring, MD, 1983.

9. Boehm, Barry W., Software Engineering Economics, Prentice Hall, Inc., Englewood Cliffs, NJ, 1981.

10. McCall, James A., and Herndon, Mary A., "Software Maintenance Management," NBS Special Publication 500-129, National Bureau of Standards, Washington, DC, 1985.

11. Lientz, Bennet P., and Swanson, E. Burton, Software Maintenance Management, Addison-Wesley, Inc., Menlo Park, California, 1980.

12. Boger, Dan C., and Lyons, Norman R., A Productivity Enhancement Study of the PMSO Software Effort, Naval Postgraduate School, Monterey, California, 1983.

13. Myers, G. J., The Art of Software Testing, John Wiley and Sons, Inc., New York, 1979.

14. Branch, M. A., Frankel, E. C., Jackson, M. C., and Laviolette, M. D., "Software Maintenance Management," Proceedings of the Conference on Software Maintenance - 1985, IEEE Computer Society Press, Washington, DC, 1985.

15. "Guidelines for Documentation of Computer Programs and Automated Data Systems," Federal Information Processing Standards Publication 38, National Bureau of Standards, Springfield, Virginia, 1976.

16. Deutsch, Michael S., Software Verification and Validation Realistic Project Approaches, Prentice-Hall Inc., Englewood Cliffs, NJ, 1982.

17. "Guidelines for Life-cycle Validation, Verification, and Testing of Computer Software," Federal Information Processing Standards Publication 101, National Bureau of Standards, Springfield, Virginia, 1983.

18. Wallace, D. R., "The Validation, Verification, and Testing of Software: An Enhancement to Software Maintainability," Proceedings of the Conference on Software Maintenance - 1985, IEEE Computer Society Press, Washington, DC, 1976.

19. Thadhani, A. J., "Factors Affecting Programmer Productivity During Application Development," IBM System Journal, Vol. 23, No. 1, 1984.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Curricula Officer, Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
4.	Professor Carl R. Jones, Code 54Js Naval Postgraduate School Monterey, California 93943-5000	1
5.	Assistant Professor Tung Bui Code 54Bd Naval Postgraduate School Monterey, California 93943-5000	1
6.	LCDR B. Wayne Silver 96 Knollsbrook Drive Nepean Ontario, Canada K2J1L8	2
7.	Commander, DMRS National Defence Headquarters Ottawa, Canada K1A 0K2	1

END

DTIC

8-86