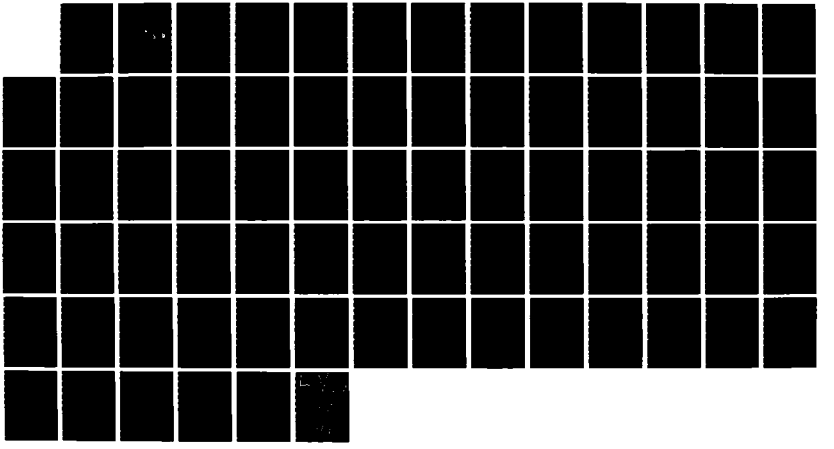
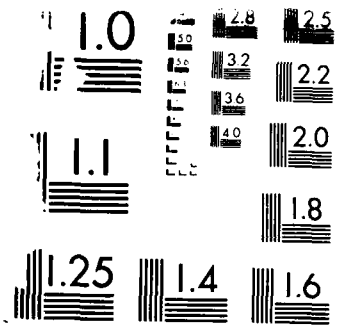
AD-A175 311 THE ROLE OF SOFTWARE DEVELOPMENT STANDARDS IN REQUIREMENTS ANALYSIS AND DESIGN(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA M Q LYLE SEP 86 1/1

UNCLASSIFIED F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

AD-A175 311

(2)

NAVAL POSTGRADUATE SCHOOL

Monterey, California

DTIC
ELECTE
DEC 2 3 1986
S D
D

THESIS

THE ROLE OF SOFTWARE DEVELOPMENT STANDARDS IN REQUIREMENTS ANALYSIS AND DESIGN

by

Margaret Queen Lyle

September 1986

Thesis Advisor: Barry A. Frew

Approved for public release; distribution is unlimited

DTIC FILE COPY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A177 311

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	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)
THE ROLE OF SOFTWARE DEVELOPMENT STANDARDS IN REQUIREMENTS ANALYSIS AND DESIGN

12 PERSONAL AUTHOR(S)
Lyle, Margaret Q.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1986, September	73

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Software Standards Software Development

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

Software is the most expensive aspect of computer systems. It also has the potential to have the greatest adverse impact on the system. This thesis examines the role of software standards in the early development phases of requirements analysis and design. Both the costs and benefits associated with the use of standards are evaluated. Tools and techniques that support the use of standards are identified and evaluated for use in producing software that is usable and maintainable. Current Navy software development guidelines are identified and evaluated with respect to current industry practices. The analysis indicates that software standards are essential in the development life cycle. Navy guidelines do mandate the use of such standards in the development of mission critical computer

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL LCDR Barry Frew	22b TELEPHONE (Include Area Code) (408) 646-2924 22c OFFICE SYMBOL Code 54Fw

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

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

#19 - ABSTRACT (CONTINUED)

software. The importance of frequent reviews and the use of supporting tools and techniques is emphasized.

Approved for public release; distribution is unlimited.

The Role of Software Development Standards
in Requirements Analysis and Design

by

Margaret Queen Lyle
Lieutenant, United States Navy
B.A., College of Great Falls, 1972
M.S., Florida State University, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
September 1986

Author: Margaret Queen Lyle
Margaret Queen Lyle

Approved by: Barry A. Frew
Barry A. Frew, Thesis Advisor

Kenneth J. Euske, Second Reader

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

Kneale T. Marshall,
Dean of Information and Policy Sciences

ABSTRACT

Software is the most expensive aspect of computer systems. It also has the potential to have the greatest adverse impact on the system. This thesis examines the role of software standards in the early development phases of requirements analysis and design. Both the costs and benefits associated with the use of standards are evaluated. Tools and techniques that support the use of standards are identified and evaluated for use in producing software that is usable and maintainable. Current Navy software development guidelines are identified and evaluated with respect to current industry practices. The analysis indicates that software standards are essential in the development life cycle. Navy guidelines do mandate the use of such standards in the development of mission critical computer software. The importance of frequent reviews and the use of supporting tools and techniques is emphasized.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 7

II. THE NEED FOR STANDARDS ---------- 10

A. BENEFITS OF STANDARDS ---------- 12

B. COSTS OF STANDARDS ---------- 14

III. NAVY GUIDELINES FOR SOFTWARE DEVELOPMENT ------- 18

A. DOD-STD-2167 ---------- 19

B. SECNAVINST 5231.1B ---------- 21

C. SECNAVINST 5233.1B ---------- 21

D. SECNAVINST 5230.8 ---------- 22

IV. STRUCTURED DEVELOPMENT TECHNIQUES ---------- 24

A. STRUCTURED ANALYSIS ---------- 26

B. TOP-DOWN METHODOLOGIES ---------- 29

C. STRUCTURED DESIGN ---------- 31

D. STRUCTURED PROGRAMMING ---------- 31

V. AUTOMATED TOOLS AND THE DEVELOPMENT ENVIRONMENT ---------- 35

VI. DEVELOPMENT TOOLS ---------- 41

A. FACILITATED APPLICATION SPECIFICATION TECHNIQUE ---------- 41

B. AUTOMATED DESIGN TOOLS ---------- 42

C. DATA DICTIONARY ---------- 47

D. HARDWARE/SOFTWARE MONITORS ---------- 48

E. TOOLS AND STANDARDS ENFORCEMENT ---------- 50

VII. STANDARDS AND THE REVIEW PROCESS ---------- 51

QUALITY INSPECTED 1

Availability Codes
Dist Avail and/or Special
A-1

VIII. CONCLUSIONS ---------- 59

A. STANDARDS DEVELOPMENT ---------- 59

B. IMPLEMENTATION ISSUES ---------- 61

C. CREATING A DEVELOPMENT ENVIRONMENT --------- 63

LIST OF REFERENCES ---------- 66

INITIAL DISTRIBUTION LIST ---------- 72

I. INTRODUCTION

Software, the fastest growing segment of the computer and information processing industry, is costly--costly to develop and even more costly to maintain. Pentagon expenditures for mission critical software totaled $11 billion last year, and it is predicted that "by 1990 the amount will more than double, accounting for roughly 20% of everything the Pentagon spends on weapons" (Newport, 1986, p. 133). Newport (1986) also reports that industry-wide, "75% of the time, businesses never use the software programs they undertake, either because they never complete them or because they arrive too late" (p. 132).

These circumstances indicate that the traditional method of developing software is not working. The current trend in software development is transforming the development life cycle from a seemingly haphazard, trial and error process into a discipline, based on standard practices, methodologies and rigorous management control (Newport, 1986). The past five years have seen major revisions of Navy software development instructions, incorporating accepted industry practices into the development of Navy software.

At first glance, the benefits of standardization seem obvious. Yet considerable time and effort are required to

implement and sustain a standards-based development environment. An effective software standards program is two-fold in nature--standards must not only be developed, they must also be adhered to and their use enforced.

Well-founded standards go hand-in-hand with an effective systems methodology (Ross, 1976). "Absence of standards makes programs difficult to maintain and impedes the development effort--particularly where a number of programmers must work together toward a common goal" ("Imposing," 1985, p. 1).

Software standards are vital in ensuring on-time delivery of quality software products and in minimizing maintenance costs. Tools, techniques and methodologies are the cornerstones of software development and maintenance. In support of a formal life cycle, software development tools are part of an emerging technology, with front-end design deemed the key to successful end products (Forman, 1980). By focusing on development, where errors are less expensive to correct, maintenance problems can be reduced (Mazzucchelli, 1985).

This thesis focuses on the requirements analysis and design phases of software development. Although development standards are equally important in later phases such as coding and testing, those areas are addressed only as they relate to the earlier stages. Second, the thesis assumes the existence of development standards, although key areas

where standards are required as highlighted. Third, consideration of development tools has been limited to those that are currently available commercially. Fourth, existing DOD guidelines for software development have been examined at the Department of the Navy level. More general guidelines, such as Federal Information Processing Standards (FIPS) publications, are not specifically addressed.

Specifically, this research is concerned with the controls and management issues that contribute to manageability of the software development process. Chapter II evaluates the need for standards, and weighs the costs and benefits associated with a standards program. Chapter III surveys Navy guidelines for software development. Chapter IV evaluates the methodologies mandated for use in the Navy guidelines and assesses their effectiveness in developing usable and maintainable software. Chapter V develops general requirements for automated tools to support software requirements analysis and design. Chapter VI surveys four specific techniques and tools that can facilitate the development process. Chapter VII details the review process which is critical to the successful use and enforcement of software standards. Chapter VIII summarizes key issues in developing, implementing and using software standards and recommends specific steps for creating a standards-based software development environment.

II. THE NEED FOR STANDARDS

This chapter establishes a rationale for developing and using standards to guide the development of software. First, the costs associated with error-laden software are identified. Second, the emphasis is focused on the earlier stages of software requirements analysis and design. Third, both the benefits and costs associated with standards development are discussed. Finally, an assessment is made concerning the use of standards in developing software.

Software is the most expensive component of computer systems, and it has the potential to have the greatest adverse impact on the system. A misunderstanding of the user's requirements and faulty debugging have far-reaching effects (Ramamoorthy, Prakash, Tsai, & Usuda, 1984). An examination of the software development process reveals that all too often, projects are delivered behind schedule, software quality is poor, the final product is cumbersome and expensive to maintain (Pressman, 1982).

One study indicates that "error removal constitutes up to 40% of the cost of a system--and that between 45% and 65% of these errors are made in system design" (Rush, 1985, p. ID/11). Unfortunately, "as errors move through the development cycle undetected, the cost to correct them increases up to multiples of 100 or more" (Mazzucchelli,

1985, p. 81). By focusing on improved development efforts, maintenance problems can be reduced.

Traditionally, software development has been viewed as an art, with few formal rules guiding the process (Frank, 1983). More recently, the concept of software engineering has evolved, seeking to bring order to the development of computer software by devising formal techniques for software development (Pressman, 1982). Based on documented standards and methodologies, "these techniques deal with software as an engineered product that requires planning, analysis, design, implementation, testing, and maintenance" (Pressman, 1982, p. xv).

A wide spectrum of software development philosophies exist, ranging from no rules at all to formal standards that are rigidly enforced. Forman (1980) concludes that the traditional method of developing software has become cumbersome and too costly for today's marketplace. With the traditional method, considerable time was spent developing specifications and code and little feedback was provided to the user until late in the development cycle. Software engineering, on the other hand, requires user involvement throughout requirements analysis and design, thus helping to produce a software product that does indeed meet the needs of the user.

A. BENEFITS OF STANDARDS

According to Boehm (1981), greater emphasis should be placed on the earlier phases in the software life cycle, as requirement and design errors are about 100 times more expensive to correct than implementation errors. With 40% of the total life cycle cost of software attributable to development and 60% to maintenance, the implementation and enforcement of software standards are major factors in reducing the costs of producing and maintaining software (Pressman, 1982). Identifying the constraints, objectives, design tools, and parameters in a standardized way yields considerable progress in dealing with problems effectively (Tausworthe, 1978). As "a major portion of maintenance activity comes from misunderstanding the user's requirements or from faulty debugging during operation, . . . some of these maintenance problems could be reduced if more attention were paid to development" (Ramamoorthy et al., 1984, p. 193).

The overall quality of software products can be improved by standardizing the practices of programmers during the entire life cycle of the product. "The subject of software standards is normally greeted with yawns of boredom or screams of anguish--yawns when the standards affect someone else, screams when they are applied to one's own project. Yet standards are fundamental to the success of most software projects" (Poston, 1984b, p. 94). Poston (1984c)

observes that several situations occur when standards are not required:

> First, the process of communicating a design to another project member requires two efforts: an initial effort to explain the technique used in creating and documenting the design, and a second effort to explain the design itself. If a standard technique is used, the initial explanation can be omitted. Second, without a standard design technique, designing a fix for a fault (bug) takes longer. The person trying to find and eliminate the fault must know not only what is wrong with the code but what it was meant to accomplish in the first place. This requires understanding the original designer's intent and, therefore the design technique. (pp. 95-96)

It is reasonable to assume that consistent, documented terminology and project standards improve communication among team members and result in fewer misinterpretations (Poston, 1984c). Standards serve as a "written, usable formalization of experience--successful experience. Their use overcomes a common problem: most project experience is lost, or at best handed down by word of mouth or individual behavior" (Braverman, 1979, p. 81). Thus, standards "reduce the vulnerability of the project to personnel turnover and time lost getting new personnel up to speed" (Peters, 1981, p. 103).

Overall benefits accrue by adopting and enforcing programming standards. The goals of standards are many--good schedule and cost performance, high product reliability, adequate documentation, increased productivity, smooth development and delivery, higher quality software, machine independence, more productive work force, and reduced production and maintenance costs (Tausworthe, 1978).

It is critical to note that "standards are not an end in themselves, but only a means to an end" (Hall, 1983, p. 112). Standards themselves do not accomplish--people do (Tausworthe, 1978).

B. COSTS OF STANDARDS

"It seems intuitively that systematic development procedures would lead to better results" (W. Myers, 1978, p. 374). Yet the development and enforcement of standards exhort a toll on the user organization. Considerable time and effort are required to develop and enforce software standards. Further effort is required to keep the standards up to date. A significant number of organizations that have standards do not enforce use of the standards:

> A [1983] survey conducted by the University of Maryland reflects that most companies have a software development policy and many of them have a "Standards and Practices" document. Of those companies surveyed, most of the military-industrial companies have a methodology manual, but unfortunately it was reported either out of date or currently "Under Revision." In addition, application and enforcement varied across projects and most standards were not enforced or augmented by software tools.
> (Thayer, 1984, p. 154)

Use of a methodology and software engineering tools requires the user to be organized and to impose certain discipline. In those organizations that do have established standards, the focus is most often on code development or tape formats (Poston, 1984c). Yet, "the characteristics being controlled are of somewhat less importance to the quality of the final software product than are the proper

development of requirements and design" (Branstad & Powell, 1984, pp. 75-76).

There are few published statistics on the costs or savings associated with having or not having standards in place on a software project. A cost-benefit analysis (Boehm, 1981) can be used in determining the potential cost savings to be realized from a standards development effort. However, at best, this estimate is based on subjective values. Fostel (1981) assesses that "accurate life cycle analysis is hard. Generally, the results will be an over estimate of the expected gains to be derived from adherence to a 'single' standard" (p. 127).

With conservative estimates, the cost of developing standards can be amortized over a number of projects. In conducting a cost benefit analysis the organization must also account for the fact that standards are not static, but rather evolve with use and advances in technology (Boehm, 1981). Thus, total costs include not only initial development efforts, but also should include costs for keeping the standards up to date.

Some organizations do not invest in imposing and enforcing the use of standards, based on a belief that such development efforts are not cost effective (Thayer, 1984). Those organizations that do have an active standards program report little savings in the short run, with a sizable overhead investment in costs for training and automated

tools acquisitions/development. "The most significant economic return on the investment in software standards comes during the maintenance phase" (Branstad & Powell, 1984, p. 76). Standardized development leads to maintainable software:

> Many maintenance problems would be solved if software were developed according to precise methodologies. Formal requirement and design specifications, detailed and clear documentation, and extensive testing and validation produce economies in the maintenance phase. These preventive maintenance activities coincide with the development activities, and better development translates as reduced maintenance effort after the delivery. (Ramamoorthy et al., 1984, p. 200)

Commitment from management, developers and users is essential for the software standards effort to come to fruition and to be effective. Each of these groups reaps the potential benefits of software standards: decreased variability, increased product quality, increased worker productivity, facilitated communication, and better control (Branstad & Powell, 1984). Thus standards should "grow out of successful, documented experience and a commitment by management to maintain a successful environment" (Braverman, 1979). But "an unused standard is worse than no standard at all" (Fostel, 1981, p. 128), providing a zero return on the standards development effort.

The evidence indicates that in the final evaluation, standards, if used, do contribute to better quality software. Standards "can be used to ensure that each and every module in a system, the overall architecture of the

system, and the decisions which lead to this configuration are all established and documented at central checkpoints during the design effort" (Peters, 1981, p. 103). Such a philosophy will result in software that both meets the requirements of the user, and is maintainable throughout its lifetime. For these reasons, the benefits of software standards offset the investment required.

III. NAVY GUIDELINES FOR SOFTWARE DEVELOPMENT

This chapter examines current Navy directives for software development. The specific guidance mandated for all phases of development of mission-critical software is discussed. The chapter concludes with an evaluation of the use of standards in developing Navy software.

A 1980 Government Accounting Office (GAO) report found that "current Government-wide ADP policy, guidance, and standards do not specifically address development, use, and evaluation of software tools and techniques" (Comptroller, 1980, p. 27). Since that time at least four Department of the Navy instructions governing software development standards have been revised and do indeed mandate the use of software tools and techniques (DOD-STD-2167, SECNAVINST 5000.1B, SECNAVINST 5230.8, and SECNAVINST 5231.1B). For example, DOD-STD-2167 issued in June 1985, "incorporates practices which have been demonstrated to be cost-effective from a life cycle perspective, based on information gathered by Department of Defense (DOD) and industry" (p. iii/iv). Emphasizing the iterative nature of software development, "the standard accommodates alternative design methodologies, the effective use of prototyping in the software development process, and the use of reusable software modules where applicable" (Heffernan, 1985, p. 16).

Navy software development standards include comprehensive development and documentation requirements, with a primary emphasis on formal methodologies and supporting software tools. The majority of Navy software directives focus on software development for mission-critical systems, with stringent guidelines mandated. Although not required for other software development projects, use of these standards is encouraged for all software projects. The general framework of these standards can be tailored to any software project, thus formalizing the development process for both mission-critical and non-mission-critical computer system software.

A. DOD-STD-2167

DOD-STD-2167 superseded DOD-STD-1979A (Navy) and was issued as part of DOD's software initiative for the 21st century. Its use is mandated in the development of mission-critical computer software (SECNAVINST 5000.1B, 1983). Based on an integrated structured approach to software development, DOD-STD-2167 (1985):

> establishes a uniform software development process which is applicable throughout the system life cycle. The software development process defines development activities which result in: (1) the generation of different types and levels of software and documentation, (2) the application of development tools, approaches, and methods, and (3) project planning and control. (p. iii/iv)

Development standards are based on a six-phase model: requirements analysis, preliminary design, detailed design, coding and unit testing, computer software component

integration and testing, and computer software configuration item testing. The standard requires the developer to "establish and implement a complete process, including methodologies and tools for developing the software and its documentation" (DOD-STD-2167, 1985, p. 11). The use of a number of structured software engineering methods is required: top-down design, modular decomposition, software development library, structured requirements analysis tools, formal and informal reviews, program design language, and structured programming. Data item descriptions for documentation deliverables are identified for each development phase.

DOD-STD-2167 provides guidelines for tailoring its application to smaller projects. By utilizing structured development techniques coupled with frequent formal and informal reviews and audits, the standard

> provides increased and more accurate visibility into the software development process, promotes earlier detection and elimination of software errors, emphasizes establishing a complete, agreed-to, understandable, and testable set of requirements prior to beginning design . . . [It is predicted that] DOD will realize an estimated $40 million savings per year through improved contractor productivity and the elimination of redundant paperwork.
> (Sprague, Maibor & Cooper, 1985, p. 48)

DOD-STD-2167 relies solely on reviews, formal and informal, to monitor conformance to development standards and to verify that the evolving software meets the requirements specifications of the user. Automation of the manual tasks of review and audit is not addressed.

B. SECNAVINST 5231.1B

SECNAVINST 5231.1B details application of the five phase life cycle management cycle to the development of information systems. The five phases are: mission analysis and project initiation, concept development, definition and design, system development, and deployment and operation. Again, the use of structured techniques such as top-down design, design walkthroughs, and program libraries is required. Conducting walkthroughs and reviews helps to monitor conformance to standards, while also helping to gauge how well the resulting requirements and design are meeting the needs of the user.

COMNAVDAC has a draft instruction providing detailed implementation guidelines of the life cycle management phases. DOD-STD-2167 relates the system life cycle to its software development phases, thus providing an integrated view of life cycle management for software development.

C. SECNAVINST 5233.1B

While DOD-STD-2167 addresses the requirements for a development methodology, SECNAVINST 5233.1B provides detailed documentation requirements. SECNAVINST 5233.1B (1979), which applies to all Navy components, including contractors, provides "necessary instructions and policy guidance for the preparation of automated data system (ADS) documents applicable to the Department of the Navy" (p. 1). The instruction prescribes use of a "standard method to

describe, format, and document data independent of any programming language" (p. 53). SECNAVINST 5233.1B primarily represents guidelines for the physical preparation of software documentation, e.g., margins, paper stock, document numbering. Although actual document content is not addressed, the contents must describe the development process as dictated in DOD-STD-2167.

D. SECNAVINST 5230.8

SECNAVINST 5230.8 directs that information processing standards be developed within Navy commands, with COMNAVDAC responsible for initiating and managing technical standards development programs Navy-wide. The Navy program is part of a larger DOD-wide effort, whose scope "includes areas such as terminology, problem description, programming languages, systems documentation, ADP equipment characteristics, input and output format and codes, source data media and fonts, systems software, . . . and teleprocessing interfaces" (SECNAVINST 5230.8, 1982, p. 2).

In endorsing the use of structured software development techniques, the Navy is supporting development of software that has been developed with greater attention given to user requirements and the maintainability of the software. The initiative not only acknowledges the need for standards for software development, but also places the requirement on Navy commands to use such standards. However, the standards will not be developed overnight. Nor will the mere

existence of standards ensure maintainable software. However, standards facilitate documentation of the complex task of software development, and as such, promote production of software that is usable and maintainable. The next step is for the Navy to require use of development standards such as DOD-STD-2167 for all Navy software, regardless of application system type.

IV. STRUCTURED DEVELOPMENT TECHNIQUES

This chapter examines the structured development methodologies mandated for use in mission-critical Navy software development projects. Strengths and weaknesses of representative structured methodologies are assessed. In conclusion, the structured techniques are judged to be effective in developing usable and maintainable Navy-developed software.

The Navy guidelines discussed in Chapter III provide a general framework within which to develop software. Structured techniques offer well-defined methods for use throughout the software life cycle of planning, development, and maintenance (Pressman, 1982). As a whole, the techniques contribute to an overall guiding methodology from system conception to design, coding and testing. The benefits to be derived from integration of the structured techniques warrant their use in Navy software projects.

Standards bound a development methodology, providing measurable milestones with which to gauge conformance. With the structured methods, specific milestones and deliverables can be identified for each development phase. DOD-STD-2167 provides detailed identification of such deliverables, in the form of its data item descriptions. Reviews, both during and concluding each phase, monitor conformance to the

standards. Pressman (1982) observes that "reviews are the only known mechanism for management and technical control" (p. 26).

DOD-STD-2167 implies that software developed under its guidelines will meet the requirements of the user. However, the standard does not specifically address measuring the impact of standards on the development process, nor is the issue of how to measure actual software quality quantified. Different measures, assessing such issues as productivity, perception, product characteristics, and impact of the software on the process, are available for such use (Sprague & Carlson, 1982).

DOD-STD-2167 provides a broad framework for software development, leaving application specific details to the discretion of the user. For example, the use of a program design language is required, although a specific language is not identified. A development methodology, consisting of methods, procedures, techniques, and tools, must be specified to each environment, marrying local needs and goals with Navy standards. The chosen methodology "must be usable as well as adaptable. It must conform to the needs, structure and . . . mission of the organization" (Levine, 1985, p. 72). Customizing to specific needs promotes both acceptance and use of the adopted methodology within the organization. Concessions must be accommodated on both sides for success to be achieved.

While many design methodologies are in use today, no single methodology can be identified as "best" in all situations. The structured programming objectives of readability, reliability, and programmer efficiency coincide with Navy software objectives (Jensen, 1981). The following discussion focuses on the advantages and disadvantages of representative methodologies, including structured analysis, top-down design and implementation, structured design, and structured programming.

The structured techniques are "consistent and rely on a simple set of rules" ("Imposing," 1985, p. 2), thus facilitating their use in many different programming environments. A 1980 GAO report concluded that:

> structured programming produces computer programs which are easier to test, and once tested, easier to modify. Thus structured programming can both reduce the chances of errors in the user results . . . and make it easier and quicker to respond to future user requests for modifications. (Comptroller, 1980, pp. 13-14)

However, no amount of testing can guarantee 100 percent software reliability. Testing can reduce "doubts and risks about the performance of the product in the target environment" (Pressman, 1982, p. 293). At best, successful testing provides "reasonable" assurance that software will perform as required.

A. STRUCTURED ANALYSIS

Structured analysis involves the use of graphic documentation tools to produce detailed specifications of the

proposed system. The primary tools of structured analysis, which serve as vehicles of communication, are data flow diagrams, the data dictionary, and structured design languages. Both data flow diagrams and the data dictionary present a top-down definition of complex data elements, thus simplifying these elements into more manageable elements (Yourdon, 1979).

Structured analysis results in partitioned designs, graphically depicted in successively detailed levels, and is implementation-independent of the resulting end-system (Yourdon, 1979). "Good requirements are complete, consistent, testable, traceable, feasible, and flexible. By just stating necessary boundary conditions, they leave room for tradeoffs during system design" (Ross & Schoman, 1977, p. 7).

Some systems are developed with no written user specifications. Although there is no absolute guarantee that structured analysis will result in what the user wants, the evolving software design is based on a formal, written assessment of user requirements. Hence, analysis is based less on intuition and more on a formal procedure for identifying and documenting user requirements. Thus, the probability of producing a system that does indeed serve a useful purpose is increased ("Structured," 1985).

As a problem solving activity, structured analysis "contributes to the accurate and detailed analysis long

before a line of code is written" (Mazzucchelli, 1985, p. 77). Ross and Schoman (1979) conclude that "use of well-structured models together with a well-defined process of analysis . . . does provide a strong foundation for actual system design" (p. 12).

Potential problems with structured analysis result from its reliance on communication between humans to derive the logical structure of the system. "Communication between two human beings always involves some risk of a misunderstanding of one sort of another" (Yourdon, 1979, p. 55). The use of diagrams serves as a communication tool, easing the potential for misunderstanding. However, "structured programming cannot resolve communication failures caused by deficient specifications or a poor development organization" (Jensen, 1981, p. 32).

Data flow diagrams "provide an easy, graphic means of modeling the flow of data through a system--any system, whether manual, automated, or a mixture of both" (Yourdon, 1979, p. 39). Complex diagrams are broken into successively simpler levels, until the lowest level of decomposition is reached. The data flow diagram focuses on the logical flow of data to derive software structure. The resulting "design representations form the basis for all subsequent development work" (Pressman, 1982, p. 202).

The data dictionary is "an organized collection of logical definitions of all data names that are used in the data

flow diagram" (Yourdon, 1979, p. 41). Every data element is defined to its lowest level of detail. The data dictionary records "all decisions related to the structure and implementation of every file and record. This information [should be] recorded in such a way that it can be easily retrieved and analyzed" (Howden, 1982, p. 320).

Structured design languages are used to describe the "bottom-level processes in the bottom-level data flow diagrams" (Yourdon, 1979, p. 42). Structured design languages describe what a module is to accomplish, without specifying how the intent or any implementation details will be achieved. They are independent of the high level programming language used in the actual coding process and result in a mini functional specification for each bottom-level process. Also known as pseudocode or program design language, structured design languages are well-organized and precise and can be written quickly and easily. They also provide "an easy-to-read overview of the procedural logic for the maintenance programmer" (Yourdon, 1979, p. 152).

B. TOP-DOWN METHODOLOGIES

Top-down methodologies facilitate management's ability to monitor and control system development (Mazzucchelli, 1985). These techniques allow for "iterations within the conventional boundaries of analysis, design, and programming" (Yourdon, 1979, p. 56). They provide a gradual progression to levels of greater and greater detail

(Bergland, 1981). Three aspects of top-down techniques can be identified: top-down design, top-down coding, and top-down testing or implementation:

- top-down design: a design strategy that breaks large, complex problems into smaller, less complex problems--and then decomposes each of those smaller problems into even smaller problems, until the original problem has been expressed as some combination of many small, solvable problems.

- top-down coding: a strategy of coding high-level, executive modules as soon as they have been designed--and generally before the low-level, detail modules have been designed.

- top-down testing or top-down implementation: a strategy of testing the high-level modules of a system before the low-level modules have been coded--and possibly before they have been designed. (Yourdon, 1979, p. 59)

Top-down design "provides an organized method of breaking the original problem into smaller problems that we can grasp, and that we can solve with some degree of success" (Yourdon, 1979, p. 62). Top-down testing and implementation facilitate major interfaces being "exercised at the beginning of the project . . . [so that] users can see a working demonstration of the system" (Yourdon, 1979, pp. 63-64) early in the development process. The top-down techniques focus

> on the overall functions and objectives of the system rather than on lines of code and concentrate on basic design characteristics required by the user. This emphasis results in a more logical, segmented development process and provides the framework for many of the leading structured methodologies that are used today. (Mazzucchelli, 1985, p. 84)

An alternative is to integrate the software from the bottom up. This approach is appropriate when "processing at

low levels in the hierarchy is required to adequately test upper levels" (Pressman, 1982, p. 300). The relative advantages of top-down versus bottom-up testing are often argued. Selection of the "best" approach is driven by software characteristics. "In general, a combined approach that uses the top-down approach for upper levels of the software structure, coupled with a bottom-up approach for subordinate levels, may be the best compromise" (Pressman, 1982, p. 302).

C. STRUCTURED DESIGN

Structured design simplifies system design by decomposing complex programs into small, relatively independent functional modules. By minimizing connections between modules (coupling) and maximizing relationships within modules (cohesion), complexity is reduced (Stevens, Myers & Constantine, 1974). The functional modules are black box in nature, performing a specific function "with little regard for the internal logical structure of the software" (Pressman, 1982, p. 292). Structured design reduces the effort required to modify programs and reduces original errors as the problem at hand is simplified (Stevens et al., 1984).

D. STRUCTURED PROGRAMMING

Structured programming is based on the principle that all programs can be written using combinations of a limited

number of logical constructs: sequence, if-then-else, and do-while. This technique minimizes the "complexity of program flow and keeps each element of a program manageably small" (Pressman, 1982, p. 131). These constructs are black box in nature, having single entry and exit points. Such an application allows code to be read and understood from the top down (Yourdon, 1979). Use of the structured constructs "reduces program complexity and thereby enhances readability, testability, and maintainability" (Pressman, 1982, p. 244).

Potential problems with the structured techniques revolve around programmer acceptance of these techniques and understanding their application. These techniques represent a change in the way programs are developed, with more formal approaches to making and documenting decisions in all phases of development. Thus programmer training and its associated learning curve must be accounted for in the overhead associated with a standards development effort.

"Structured programming--improperly applied--is no better than traditional methods of program design" (Jensen, 1981, p. 32). It is still very possible to write poor code using the structured techniques. "Even the best structured programming code will not help if the programmer has been told to solve the wrong problem, or, worse yet, has been given a correct description, but has not understood it" (Ross & Schoman, 1977, p. 6). Overall, Navy software

standards do not address this issue, implying that the use of structured programming is the panacea for most development problems.

The use of structured techniques helps reduce errors in analysis and design, thus reducing the costs of testing and maintenance. A Computer Sciences Corporation survey on the use of structured analysis and design techniques over a seven year period revealed that almost 50% of the development time is spent on analysis and design when structured techniques are utilized, as opposed to 30% when structured techniques are not employed (Mazzucchelli, 1985).

Manual approaches to structured analysis are prone to costly errors. Problems include the time required to redraw diagrams for every revision, the volumes of paper to shuffle, and the number of errors inherent in any human process. The development of automated tools to support structured analysis is contributing to their efficiency:

> While contributing to better communication and organization, structured techniques--in and of themselves--do not solve the productivity/quality crisis. The automation of the software development process has begun to address these problems during the last few years. Tools are now available to automate the job of software engineering. A variety of tools can be chosen to address different development functions and contribute significantly to increased productivity and especially to quality. (Mazzucchelli, 1985, pp. 80-81)

W. Myers (1978) concludes that "modern programming practices are effective in improving the processes of software development" (p. 384). By automating some of the functions associated with the structured techniques, their

effectiveness is further increased. The next chapter conceptualizes the use of automated tools in enforcing software standards.

V. AUTOMATED TOOLS AND THE DEVELOPMENT ENVIRONMENT

This chapter establishes the rationale for using automated tools in support of software development. First, the elements of a tool-supported environment are defined. Second, the costs and benefits, both long and short term, associated with the use of automated tools are discussed.

As Blum (1985) observes, "the software community has done an excellent job of attempting to automate everyone's job except their own" (p. 43). Software design is still all too often a paper and pencil drill, with redesign a major effort of juggling plastic templates and mountains of paper. However, a well constructed tools environment can systematize and improve the software development process, bringing increased standardization and automated control (Federal Software, 1982).

An integrated development environment should encompass five elements: tools, standards, procedures, training, and control measures (Federal Software, 1982). Although all are essential to the success of a project, the focus here is on control measures and the relationship between tools and standards. The successful environment uses a system development methodology in conjunction with computerized software tools. Thus, tools are needed that support

established systematic procedures, following a system from its conception to its final design (Egyhazy, 1985).

The ideal tool would provide a single solution to manage the complete life cycle, from requirements analysis to maintenance, along with an associated methodology and a working implementation (Miller, 1979). Available technologies "provide various levels of computer assistance in most or all areas of the development life cycle, at levels including requirements definition, systems design, coding, testing, documentation, and maintenance" (Gillin, 1984, p. 1). Although not yet available in any one package, "industry observers predict that manufacturers will [soon] develop complete software development product lines to provide integrated tools that together encompass the entire software development life cycle" (Mazzucchelli, 1985, p. 86).

An integrated tools environment "provides an opportunity for standardization within the production development environment" (Pfrenzinger, 1985, p. 44). To be most effective, the environment should yield designs and code consistent to the same level of detail. "If a design is expressed in a consistent fashion, then some measure of its contents can be made" (Brown, 1985, p. 135).

Higher quality software can be achieved through the use of computer technology. A 1980 GAO report advocates the use of automation and identifies a number of benefits that

software tools and techniques can offer the federal government:

- better management control of computer software development, operation, maintenance, and conversion;
- lower costs for computer software development, operation, maintenance, and conversion;
- feasible means of inspecting both contractor-developed and in-house-developed computer software for such quality indications as conformance to standards and thoroughness of testing. (Houghton, 1982, pp. 1-2)

To be effective, tools require standards, order and discipline, with their functional capabilities defined by organizational procedures and standards (Fisher & Herdt, 1985). The structure and detail required by automated aids leave less to the discretion of programmers, resulting in more consistent products. "Standards provide the means of customizing a set of tools so that they are used effectively within an organization" (Hall, 1983, p. 114).

A number of integrated tools are available, offering support for standardized development procedures and identified methodologies. The hardware suite for a development environment is a major factor in tool selection. Different products offer diverse approaches to development, ranging from mainframe to microcomputer applications.

This effort focuses on a microcomputer environment, where "by using the right software tools and by applying traditional system life cycle methodology, the PC can be a cost-effective alternative to applications development on mainframes" (Michielsen, 1986, p. 96). There are also a

number of PC-based development utilities, for such tasks as report generation and screen formatting, which further integrate the development process (Michielsen, 1986). PC-based tools are portable and the resulting products can be implemented on a variety of mainframe environments (Leavitt, 1986). Also, microcomputers are more interactive than mainframes, and "the terminals can communicate with each other and with a central source without tying up the mainframe" ("Proper," 1984, p. 17).

Automated tools speed up the development process and enforce discipline (Martin & Hershey, 1986). "Almost all software development organizations can see productivity and quality increases from their staffs with the use of appropriate automated tools" (Mazzucchelli, 1985, p. 86). Further, "automated design techniques can greatly improve the technical soundness of an installation. They provide capabilities not available to designers using manual methods. These capabilities help to reduce the life cycle costs of the system" ("Application," 1981, p. 10).

Selection of a tool is a long-term strategic decision, often representing a major investment in development, training and/or maintenance efforts. Choosing a particular automated package should be subjected to the "same economic, operational, and technical criteria used to determine application requirements" (Michielsen, 1986, p. 96).

The primary advantage of automated tools, as a whole, is their ability to graphically illustrate how data moves through a system and to allow the programmer to easily make changes to a system model (Korzeniowski, 1985). In automating the tasks of system designers, the computer can:

> process data cheaper, faster, and more rigorously than a human programmer . . . and can check a program's adherence to standards in a way that humans cannot, and will not, do . . . [things must be described] unambiguously and completely before a computer can process them . . . these can be the disciplining force that directs the project, provides a structure for analysis, and provides a basis for controlling the project. (Federal Software, 1982, p. 7)

Users report little gain in the initial use of automated tools--"all the words, symbols and layout choices have to be entered to start a design document. The advantages comes with the follow-up work, the editing, the corrections, the repositioning of paragraphs or design elements" (Leavitt, 1986, p. 59). Users do report significant time savings in later stages, such as testing and maintenance. "Often the improved quality of the specifications produced more than make up for the cost of a development tool because they save so much coding and maintenance time" (Mazzucchelli, 1985, p. 86).

The majority of current system development tools focus on the early analysis and design of a system (Inmon, 1985). Use of tools in these stages facilitates errors being "identified and eliminated more easily during the period of development where they remain inexpensive to correct"

(Mazzucchelli, 1985, p. 86). "When appropriate automation is available, it becomes easier to perform the work the standard way than by any alternative means. In such instances, standards audits or enforcement becomes transparent, since the development process incorporates the standard" (Branstad & Powell, 1984, p. 74). Thus, the use of automated tools and/or methodologies promotes consistency in product design. Supported by established software and development standards,

> applications development on micros [can] be managed and conducted most efficiently. If software tools, communication parameters, and screen-handling conventions are predefined, then the process of managing development as programmers move from application to application can be handled efficiently. The same software tools and methodologies provide a common ground of understanding to build systems. (Michielsen, 1986, p. 98)

VI. DEVELOPMENT TOOLS

Having established a rationale for the use of automated tools in support of software development, the focus now shifts to specific tools and techniques that facilitate the development process. This chapter examines four types of commercially available tools that can improve the development process--a requirements specification technique, a class of automated design tools, the data dictionary, and hardware/software monitors. These tools support the use of the structured techniques mandated for use in Navy software development. The tools can be used singly or in conjunction with each other to compound their effectiveness.

A. FACILITATED APPLICATION SPECIFICATION TECHNIQUE

Although some development processes such as statement of objectives and requirements analysis are subjective and thus more difficult to define, methods have been developed to facilitate these tasks and to help the requester conceptualize and verbalize his needs (Rush, 1985). One of the newest requirements analysis methodologies is the facilitated application specification technique (FAST) which uses a structured, trained-leader workshop to focus on the requirements definition stage of system design. FAST is actually a series of interactive design techniques which are used "to extract high-quality business system specifications

from end users in a workshop environment. They are not replacements for analytical methodologies, but they can all work with and supplement any methodology" (Rush, 1985, p. ID/13).

FAST utilizes specific techniques in a structured meeting setting to facilitate communication between system designers and end users, focusing on the interviewing and information gathering processes required to define system requirements. In the workshop, users describe "their business functions, information needs, data elements used and how they want to interface with the system" (Rush, 1985, p. ID/14). In assessing the success of the FAST techniques, Rush (1985) concludes that the "specifications developed from these methods have been more thorough, better documented and more consistent than with a less rigorous approach as well as being obtainable more quickly and at less expense" (p. ID/15).

B. AUTOMATED DESIGN TOOLS

One type of commercially available microcomputer package which can ease the tasks of software development focuses on the early stages of system definition and preliminary design, facilitating the planning of program logic before code is actually generated. These design tools automate many of the time-consuming tasks of system designers, including graphic designs, document production, word

processing, error checking/consistency, and data dictionaries (W. Myers, 1985).

These design tools automate many of the manual processes of analysis and design by offering a standardized, yet flexible way to document project components without having to maintain the entire design on paper. "Good automated design tools support an evolutionary development process that overcomes the rigidity of the classical process with its frozen specifications and long lead time between initial request and operational implementation" (Leavitt, 1986, p. 59).

These packages focus on detailing the logical flow of data through the proposed system and combine the logic of structured analysis with the graphics capability of a microcomputer (Leavitt, 1985). Hardware-wise, the design tools require a PC with 640 KB of memory and a 10 MB hard disk. Representative packages include Excelerator (Index Technology), Analyst Toolkit (Yourdon) and DesignAid (Nastec).

The design tools "help the user focus on the business unit to be served, to determine what problems it faces and how well they are being met independent of the [actual] computing environment" (Leavitt, 1986, p. 58). With a strong emphasis on diagramming aids and the data dictionary, these tools facilitate system documentation. The tools contribute to a well-organized and well-defined design phase, documenting how data moves through the system. A

design data base is defined in terms of the logical functions of the data elements, "rather than in terms of the hardware or software that use them" ("Case," 1984, p. 16).

A major strength of the design tools is their strong graphics capability. Change is an integral part of system design, and automated design packages "enable a programmer to easily depict changes to a system model" (Korzeniowski, 1985, p. 63). The tools offer a number of predefined graphics symbols, with particular emphasis on data flow and data structure diagrams. Supported by the use of a mouse, the tools are menu driven, simplifying creation, modification, and validation of design diagrams and documentation. The user can quickly and automatically construct and reconstruct data flow diagrams and structure charts. The system automatically captures process names assigned to the elements of the data flow diagrams, and maintains a consistency check throughout the process. Data flows are balanced from level to level of abstraction, and discrepancy messages are generated as required. Another feature is the capability to scale the size of the diagrams and charts and their accompanying labels and data elements.

As the design evolves, a data dictionary is dynamically created, recording data elements and where and how they are used. The dictionary serves as the central repository for information about the system, including such elements as processes, functions, screen descriptions, and data flows.

This same information is essential in later development stages such as detailed design and coding. The tools also support importing/exporting data dictionary definitions from/to other sources.

Another important feature of the design tools which assists the designer is the consistency/accuracy check. "Checking for errors represents one of the most important features from automated tools. Consistency checkers go through parts of a project manual or the entire model and check consistency between data flow diagrams, data dictionary entries and process specifications" (Mazzucchelli, 1985, p. 83). Diagrams are automatically validated for accuracy and completeness, with syntax and definition errors identified in error reports and/or on the screen. This feature also facilitates conformance to design standards, if the standards of the organization parallel those implemented with the tool. "Automating error checking reduces the total number of errors simply because it does a complete and thorough check--one more comprehensive than any analyst would be willing or able to do manually" (Mazzucchelli, 1985, p. 83).

Although initially configured to the unique structured development methodology of the tool, some of the design tools offer an optional capability which allows the user to define symbols and documentation standards. Such a

capability allows the organization to further tailor the tool to support its own design standards.

A number of utilities, which also serve to ease the job of the designer, are included with the basic tool. Screen facilities such as menus, reverse video blinking, and help messages are available. Some of the tools support free form graphics and a sketch capability which further automate tasks normally done by hand.

Automated design tools produce machine readable and easily modifiable documentation, thus facilitating documentation of the design process. Design and documentation tasks can be done online and cataloged in the design dictionary, thus producing system documentation as a byproduct of the design process. A major feature of the tools is the ability to incorporate text and graphics. As design changes are easily made, the tools promote completeness, consistency and accuracy of the design.

In assessing the benefits to be derived from the use of automated design tools, "users of the microcomputer packages admit that they do not save time in the first phase of systems development but claim time savings in later stages" (Korzeniowski, 1985, p. 63). "All the words, symbols and layout choices have to be entered to start a design document. The advantage comes with the follow-up work, the editing, the corrections, the repositioning of paragraphs or design elements" (Leavitt, 1986, p. 59).

C. DATA DICTIONARY

The data dictionary is critical to the success of any software development process. Although the data dictionary is available in both manual and automated forms, this discussion assumes an automated PC-based capability. The data dictionary is supported in both mainframe and microcomputer environments, thus facilitating transfer of data to and from both environments. Used to define both data and software design, the data dictionary controls a clear and consistent definition of data used in software design and coding. Thus, design changes can be implemented consistently and completely (Brown, 1985).

The data dictionary is the primary tool to control defining and describing data (Leong-Hong & Plagman, 1982). The dictionary

> explicitly represents the relationship among data and the constraints on the elements of a data structure. Algorithms that must take advantage of specific relationships can be more easily defined if a dictionary-like data specification exists. (Pressman, 1982, p. 230)

The data dictionary directly supports the design process and indirectly improves the development process. Data elements are defined in terms of their functions, not the hardware and software that use them ("Case," 1984). Ultimately incorporated into system documentation, the data dictionary documents a number of different specifications as they are developed and refined--data entry, file content,

report layout, data element narrative references, and file program cross references ("Software," 1985).

The data dictionary can also play an integral role in defining and enforcing data standards:

> It can be used to promulgate because the [data dictionary] can be made to record only acceptable standard data definitions. Databases or application systems requesting data entities will only be able to retrieve standard descriptions of data entities and will only have standard data names. The [date dictionary] can be used to monitor and to enforce standard data definitions because through its edit and validation facilities, it can screen out nonstandard, or nonconforming data elements. If and when nonconformance is detected, the DBA can take appropriate action. (Leong-Hong & Plagman, 1982, p. 52)

D. HARDWARE/SOFTWARE MONITORS

As noted earlier, the primary focus of this research is on the early development stages of requirements analysis and design. As such, no discussion is made of the use of tools in the later stages of coding, testing and maintenance. The next type of tool is best utilized during testing and actual use of the software product.

Once the application program is ready for use (i.e., mainframe processing), hardware and software monitors are another type of tool that can be employed in monitoring software development and conformance to standards. Although available only on mainframe computers, hardware and software monitors can be used to monitor conformance to coding standards and measure quality factors such as efficiency of systems and applications (Gitomer, 1984). In different ways, both hardware and software monitors "collect data on

system utilization during normal operations for on-line or after-the-fact analysis" ("Twelfth," 1984, p. 28). These tools provide data on workload analysis, system tuning and capacity planning (Freedman, 1985).

These monitors can identify inefficient applications, and corrective action can then be taken. Software monitors can generate data about applications program execution, such as "number of transactions processed, input/output operations executed, records added or deleted to a file or database, . . . time elapsed while the program was in execution, [and] CPU time for the program" (Gitomer, 1984, p. 52).

Although both types of monitors can be used in assessing the quality of the finished product, inherent characteristics of each one result in their capabilities not being fully exploited. Hardware monitors are expensive and complex to use ("Thirteenth," 1985). The information produced is often not worth the required investment in time and money (Gitomer, 1984). Although easier to use, software monitors have extremely large core and CPU requirements ("Thirteenth," 1985). As their use often can cause a virtual standstill of online processing, actual use is often restricted to short, infrequent time intervals. Such sporadic use decreases the potential role that software monitors can play in monitoring applications programs and their adherence to software standards (Gitomer, 1984).

E. TOOLS AND STANDARDS ENFORCEMENT

The four types of tools evaluated in this chapter support development standards and can thus improve the development process. In surveying current commercial software packages, none were found to support total automation of requirements analysis and design. At this time, technology does not support automation of the actual creative process required in these stages. The tools discussed do, however, provide support for the manual documentation of the creative process. The four types of tools also provide support for the review process in the next section, with the review being the focal point in monitoring and enforcing the use of standards.

VII. STANDARDS AND THE REVIEW PROCESS

This chapter examines the relationship between standards and a review process to monitor and enforce their use. Environmental conditions that support the use of standards are identified. Next, the role of verification and validation in the development process is defined. Finally, the concept of the review is defined and evaluated.

Once development standards are established, they must be continuously and actively monitored for proper implementation and use (Foote-Lennox, 1984). Standards that are not monitored may not be followed. "Too often no specific procedures are advocated and evaluation is based predominantly on the production of a running program. Since the process of development is largely ignored, the programmer has little motivation to be systematic" (Egyhazy, 1985, p. 8).

What is meant by conformance to standards? At the deliverable level, it means

> assuring that each intermediate product that is the output of one step in a multistep process is of the highest possible quality before it becomes the input to a succeeding step. This not only assures the quality of the next deliverable, but increases the chances of achieving it at the first attempt. (Duncan, 1986, p. 136)

As discussed earlier, there are commercially available tools that support the use of standards. These tools do not automate the task of ensuring actual conformance to

standards. The tools do, however, ease the steps of determining software requirements and both logical and physical design.

Certain conditions are necessary for a successful standards program. Conformance mechanisms ensure that the product meets requirements and promote the use of the standards. Standards which actually ease the job of designers and programmers promote their own use.

The individual designer/programmer and that individual's perception of the standards ultimately determine the success or failure of the standards program. "The largest single component affecting productivity and product quality is the individual; it has a weighting factor at least twice as large as any other" (Blum, 1985, p. 46). "A key measure of success is the degree to which the development staff view the standards as tools in their work. Participation in the selection and shaping of the standards and methods promotes this attitude and the consequent useful application of the standards" (Freeman & Hermon, 1983, p. 106).

Management commitment is also essential. Using standards often involves a change from the way software has been developed. It is often a

> major and potentially disruptive change from the status quo. . . . Any lack of management commitment is instantly visible and transmitted to the staff. Giving lip service to the adoption of standards while failing to support them with real resources is worse than doing nothing at all. (Freeman & Hermon, 1983, p. 105)

Most people resist change, as it represents a departure from the status quo. Thus, "the change, [in this case the use of standards], must be motivated. . . . An educational program should be initiated. People must understand the standards in order to use them" (Branstad & Powell, 1984, p. 74). Staff members must receive formal training to support the standard methodology adopted, and to help them integrate the standards into their work.

To effect enforcement of software standards, one approach

> concentrates on the process by which the software product is produced rather than on the characteristics of the product itself. To effect the approach, specific steps in the development process are standardized both with respect to their occurrence and to the techniques used to accomplish the step. (Branstad & Powell, 1984, p. 74)

Thus, use of a formal methodology, such as the overall structured approach detailed earlier, not only promotes use of standards, but eases the task of monitoring actual implementation. When a design is expressed consistently, some measure of its contents can be made. "Completeness and consistency can be expressed in terms of mismatched interfaces and processes, or by the data a process uses" (Brown, 1985, p. 135).

Reviews are key to the enforcement of standards. Navy guidelines mandate formal and informal reviews as control mechanisms for software quality and conformance to development standards. "Shifting attention from controlling the development process toward managing the development of a

design will provide a clear understanding of the tasks and issues involved in the development process" (Brown, 1985, p. 136). Thus, emphasis should be on the early design stages,

> since the cost for software corrections during operations is many times the cost incurred in detecting problems during design, inspections provide an unusual leveraging of cost/benefit over the entire life cycle of the software. (Frank, 1983, p. 85)

Validation and verification are two key aspects of the review process (Boehm, 1984). "Validation ensures that an implementation of a design actually behaves as the design intended. Verification determines that a design has been consistently stated and constrained throughout its life cycle" (Brown, 1985, p. 134). In order to enforce standards, "both the software product being created and the process of creating it must be measurable, repeatable and changeable. These most important requirements lead us to basic design constraints on the environment" (Poston, 1984a, p. 87). Thus, verification is part of every successful review. Both the automated design tools and the design data dictionary support the verification process with their automated cross-referencing capability.

Software must be reviewed both during and concluding each phase of development. To be effective, reviews should be based on the development methodology, standards and tools. The development methodology serves as the basis for the review process, which maps the process from stage to stage. Software design and code:

> cannot be formally inspected or reviewed without reference to basic standards. The standards represent the discriminating measure of acceptable and unacceptable design and programming practices. Standards can also serve the extremely vital purpose of insuring that the development process is not reduced to simply generating compliable code. (Frank, 1983, p. 82)

Reviews must be based on a formal plan in order to assess a product both in terms of meeting requirements and conforming to standards. Such a plan must state how quality will be examined and measured, and identifies controls used to ensure that defined standards and procedures are followed.

> Differing levels of experience, ability, style of work, and even attitude, can cause variations in quality levels within the same department. But if quality plans are mandatory and are produced according to standard guidelines, the variations in quality should diminish. . . . People rarely read standards manuals from cover to cover, but if at the start of a project they were told what the relevant standards were and exactly where they could be found, the chances of the standards being applied correctly would increase. (Duncan, 1986, p. 136)

Tools, both automated and manual, are integral to the review process. Automated design tools provide assistance during the inspection/review process. These tools track and record changes. For example, the design tools generate reports indicating failure to achieve relevant consistency and conformance to the design standards embodied in the tool. Both structural and design inconsistencies in successive levels of design are audited and flagged if not achieved.

The structured techniques produce a hierarchy of design detail. Therefore, successive levels of detail can be

checked for consistency and completeness, i.e., "for processes without inputs, data elements that are not used, and so forth. The expansion of requirements into design and design into detail design will provide some confidence that requirements can be traced to specific features" (Brown, 1985, p. 135).

Automated documentation also makes for faster and easier review. Documentation can be presented interactively, and the reviewer can assess that all functions are present, required text, data and diagrams are included, and required detail is correct. Automated documentation allows the system to be described more completely and accurately and is more concise than manual methods. "Updating is made easier because only the functions affected by the change require recalling. This can be done automatically, while with manual methods one has to review the whole lot" ("Structured," 1985, p. 507).

The review process also contributes to better documentation, as the review must have complete and comprehensive documentation to be effected (Citron, 1984). Reviews

> can serve as the basis for evaluating adherence to standards and procedures, ascertaining the quality of products, and providing the needed information for managerial decisions. If an acceptable level of quality is not attained at a given checkpoint, it is a good time to make changes or reconstruct the products and then review for quality assurance again. This will reduce the possibility of errors in one software development stage from cascading into a later stage.
> (Federal Software, 1983, p. 68)

Enforced use of standards will "reduce the overall cost of software by reducing the effort spent on maintenance through better planning, design and control of software resources" (Federal Software, 1983, p. 9). "The disciplined, structured methods some products impose on developers can improve application design and quality. These tools can enforce structured, standardized techniques on programmers. That forces the creation of a maintainable product" (Gallant, 1984, p. 26).

The review process helps define, audit and enforce standards (Yourdon, 1978). It ensures that software meets requirements and performs as intended. Although reviews take time, the time spent considerably reduces future time spent on testing, integration and maintenance (Citron, 1984). The payoff "is in higher performance in quality and delivery of the product . . . primarily because of the substantial reduction in the maintenance activity resulting from the higher quality of the product" (Frank, 1983, p. 85).

Reviews, driven by a quality plan, are the key elements in properly implementing and using software development standards. The quality plan details the standards required in producing the software. The review itself determines if the product has been produced according to standards (Duncan, 1986).

All the enforcement mechanisms in the world will not ensure conformance to standards unless the actual users of the standards are committed to their use:

> A standard is only a standardized method. Only if it in fact serves some well-defined need will it be accepted and will it in some way create a standard portion (that part specified by its applicability clause) of a total world, the majority of which will remain nonstandard. A structured network of carefully devised, bounded, and successful standards will be found to be so useful that the self-serving interests of those affected by them will cause their acceptance and adoption as a natural optimization step. There is no room for alternate possibilities. Standards cannot create a standard world.
> (Ross, 1976, pp. 596-597)

VIII. CONCLUSIONS

This chapter recaps key issues in using standards and tools in a software development environment. First, the requirement for three types of software standards is discussed. Second, key implementation issues are addressed. Finally, recommendations are made for creating a development environment.

A. STANDARDS DEVELOPMENT

Although not specifically addressed in this thesis, the development of standards must precede their use and enforcement. The following discussion provides a general overview for the actual selection and development of software standards. At the local command level, the general guidelines in DOD-STD-2167 and the other Navy software standards discussed in Section III must be specified to the local processing environment. Such standards establish quality control measures and "norms of good practice, while providing leeway for the use of diverse development techniques and approaches" (Branstad & Powell, 1984, p. 73).

Standards should be developed for all phases of the development process and can be divided into three categories: life cycle development, developer support and job function (Poston, 1984a). In each category, the standards should document successful experience and

represent the way that software products are to be developed within the organization (Braverman, 1979).

Life cycle standards detail the actual development process, from requirements analysis to coding to testing and maintenance (Poston, 1984a). DOD-STD-2167 provides the general framework upon which Navy commands can develop life cycle standards. A six phase life cycle, based on a structured approach, is mandated.

Support standards provide the framework for support to the actual users of the standards--training, tools, standards and metrics, techniques, and management policies (Poston, 1984a). The development staff is "the most important factor in determining product quality and process efficiency" (Poston, 1984a, p. 88). Thus, the support standards detail what, why and how particular staff members are to perform their jobs. Training support and the use of tools are specifically detailed in this type standard.

It is also important to partition available staff positions into job functions, the third major area for standards development (Poston, 1984a). These standards address project management, project assurance, product development, verification/validation/testing, and configuration management. Depending on the number of persons assigned to a project, one person could perform all functions on a small project, while on a large project, several persons could be assigned to each function. Most

important is clear assignment of function so that all personnel know what is expected of them.

B. IMPLEMENTATION ISSUES

Branstad and Powell (1984) address specific implementation issues in the successful establishment and use of software standards, and observe that such an implementation involves "significant understanding and insight into the state of current technology, human nature, people's ability to deal with change, and the goals of the particular organization and project" (p. 74). These issues center around the selection, introduction, support, and use of standards and are discussed below.

Prerequisite to implementing a standards program "are the existence and enforcement of definitive programming standards, as well as management's understanding, support, and trust--which itself is a function of the quality and timeliness of the work produced by the system's group" ("Structured," 1985, p. 501). Standards must be measurable in order to be enforced (Branstad & Powell, 1984). "The measurement may relate to size, complexity, functionality, or the number of errors discovered during reviews" (Poston, 1984a, p. 89). Thus it must be possible to determine if the work does comply with the standard. With such standards as module size or naming conventions, compliance is easily measurable. With quality-related standards, direct measurement is more difficult, as quality is a more

subjective judgment. The use of a standardized development methodology focuses on the way in which the product is developed, with quality achieved by controlling the development process (Branstad & Powell, 1984).

Often, the implementation of software standards represents a change in the way in which software is developed. "The development process is redefined in a number of ways, new methods and tools are introduced, and additional control is often imposed" (Wedburg, 1981, p. 134). As Oliver (1985) observes, "change carries a cost, which must be weighed against its benefits" (p. 19). Not only is management commitment essential, but actual users of the standards must also understand the proper use of the standards. "Office politics, personalities, motivation, collaboration, and performance criteria are all crucial considerations when introducing change" (Freeman & Hermon, 1983, p. 106).

The staff performing the work is the single most important factor affecting the quality of the product and efficiency of the development (Poston, 1984a). Although choosing both a methodology and supporting tools is integral to the success of a standards program, it does not guarantee success. "To actually change the behavior of systems professionals requires marketing, education, monitoring, and feedback. (And--if the feedback so indicates--more marketing, more education, more monitoring, and more

feedback until the desired change is achieved)" (Hedrick, 1986).

As Ross (1976) observed, "a standard is only a standardized method. Only if it in fact serves some well-defined need will it be accepted and will it in some way create a standard portion . . . of a total world, the majority of which will remain nonstandard" (pp. 596-597). Education, training, and continual updating of the standards will assist users in correctly applying the standards and promote the development of quality software.

C. CREATING A DEVELOPMENT ENVIRONMENT

Based on this research effort, the following steps are judged to be critical in establishing a successful software development program. These steps are achievable and can be implemented in Navy software development environments.

The first step in any standards development effort involves adoption of a life cycle development methodology. Such a methodology brings discipline to the entire development process, from requirements analysis to configuration management. In the case of mission critical Navy software, DOD-STD-2167 mandates the use of a structured methodology. Representative methodologies include Jackson, Yourdon and Warnier (Pressman, 1982). The particular methodology is not as important as the fact that a methodology is adopted. Levine (1985) reports that "nearly 800 purchased systems development methodologies or standards [are] in place at

major organizations" (p. 72). Key to the selection of a methodology is its ability to support the needs and mission of the organization. "A methodology embraces the way an organization designs, develops and implements systems" (Levine, 1985, p. 72).

Once the methodology is identified, the organization should identify tools to support the methodology. Two options are available in acquiring tools--off-the-shelf and custom developed. If commercially available software supports the requirements of the organization, such packages should be utilized:

> In contemplating procurement of any software program, the economics always favor the low-risk, high leverage solution of purchasing existing products. In make-or-buy trade-offs, paybacks from a purchase usually occur in one-third the time, while investment costs are also recaptured in one-third the time. The development time is reduced to one-fifth and development costs to as little as one-seventh. (Frank, 1983, p. 164)

Equally important is the fact that future maintenance costs are substantially reduced when off-the-shelf software is utilized (Hedrick, 1986).

A minimum set of tools is required to implement a standards program. In evaluating available design tools, several candidate packages support the different structured methodologies. At least one of the integrated design tools discussed earlier should be selected to support the life cycle methodology. Additionally, a requirements analysis technique should be employed to facilitate accurate identification of user requirements. During the design process, a

data dictionary is essential for consistent identification and use of data in both design and later in the coding process. Finally, hardware or software monitors are required to fine tune the final product for efficient operation.

Actual standards must then be developed to finely detail the methodology, its application and use. Each of the three major types of standards discussed above is required. A standards coordinator should be designated to monitor the use of standards, keep them up to date, and ensure that training is provided. Finally, the review process should be formalized with the designation of a review group to ensure not only that standards are being employed, but that standards are being correctly employed.

LIST OF REFERENCES

Application system design aids. (1981, October). EDP Analyzer, 19, pp. 1-12.

Bergland, G.D. (1981, October). A guided tour of program design methodologies. Computer, 14, pp. 13-37.

Blum, B. (1985, January). Understanding the software paradox. ACM SIGSOFT Software Engineering Notes, 10, pp. 43-47.

Boehm, B.W. (1981). Software engineering economics. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Boehm, B.W. (1984, January). Verifying and validating software requirements and design specifications. IEEE Software, 1, pp. 75-88.

Branstad, M., & Powell, P.B. (1984, January). Software engineering project standards. IEEE Transactions on Software Engineering, SE-10, pp. 73-78.

Braverman, P.H. (1979, July). Yes, folks, standards are a many-splendored thing. Computer, 12, pp. 81-84.

Brown, P. (1985, April 15). Managing software development. Datamation, 31, pp. 133-136.

Case Western implements automated design tools. (1984, August 20). Computerworld, 18, p. 16+.

Citron, A. (1984, January). A software review method that really works. BYTE, 9, pp. 437-440.

Comptroller General of the United States. (1980, April 29). Wider use of better computer software technology can improve management control and reduce costs (FGMSD-80-38). Washington, DC: General Accounting Office.

DOD-STD-2167. (1985). Software development standard documentation set.

Duncan, M. (1986, March 15). But what about quality? Datamation, 32, pp. 135-139.

Egyhazy, C.J. (1985, January). Technical software development tools. Journal of Systems Management, 36, pp. 8-13.

Federal Software Testing Center. (1983, June). Establishing a software engineering technology (SET) (Report OSD/FSTC-83/014). Washington, DC: General Services Administration.

Federal Software Testing Center. (1982, February). A software tools project: A Means of capturing technology and improving engineering (Report OSD/82-101). Washington, DC: General Services Administration.

Fisher, G. & Herdt, D. (1985, July). Establishing a programmer's workbench. Federal Software Information Exchange, 3, pp. 5-9.

Foote-Lennox, T. (1984). More standards, more trouble. In Proceedings of the Third Software Engineering Standards Application Workshop (pp. 55-58). Silver Spring, MD: IEEE Computer Society Press.

Forman, J. (1980, June). Implementing software standards. Computer, 13, pp. 67-70.

Fostel, G.N. (1981). Principles of software standardization. In Proceedings of the Software Engineering Standards Application Workshop (pp. 125-133). Silver Spring, Md: IEEE Computer Society Press.

Frank, W.L. (1983). Critical issues in software. New York: John Wiley and Sons.

Freedman, D.H. (1985, October). Performance and capacity management: No longer a technician's game. Infosystems, 32, pp. 54-56.

Freeman, P., & Hermon, R.A. (1983). Lessons learned from the development and application of software engineering standards. In Proceedings of the Second Software Engineering Standards Application Workshop (pp. 103-108). Silver Spring, MD: IEEE Computer Society Press.

Gallant, J. (1984, December 10). Do tools help find DP gold? Computerworld, 18, p. 1+.

Gallant, J. (1986, February 24). Software spending to rise. Computerworld, 20, p. 19+.

Gillin, P. (1984, August 20). Computer-aided software engineering: Automating DP. Computerworld, 18, p. 1+.

Gitomer, J. (1984, Summer). Measuring system performance with software monitors. Journal of Information Systems Management, 1, pp. 50-55.

Hall, P.A. (1983). Standards as paper tools. In Proceedings of the Second Software Engineering Standards Application Workshop (pp. 111-114). Silver Spring, MD: IEEE Computer Society Press.

Hedrick, R.T. (1986, May 15). Improving productivity at Northern Trust. Datamation, 32, pp. 97-104.

Heffernan, H. (1985, June 7). Revolutionary software standard near. Government Computer News, 4, p. 1+.

Houghton, R.C. (1982, March). Software development tools (Special Publication 500-88). Washington, DC: National Bureau of Standards.

Howden, W.E. (1982, May). Contemporary software development environments. Communications of the ACM, 25, pp. 318-329.

Imposing programming standards. (1985, October). Datapro Management of Applications Software, 7, pp. 1-3.

Inmon, W. (1985, March 4). Systems tools: Let buyer beware. Computerworld, 19, p. 45.

Jensen, R.W. (1981, March). Structured programming. Computer, 14, pp. 31-48.

Korzeniowski, P. (1985, May 27). Micro tools speed system development. Computerworld, 19, p. 55+.

Leavitt, D. (1986, February). Design tools: The real starting point. Software News, 6, pp. 57-59.

Leavitt, D. (1985, February). The proper design tools can bring improved productivity. Software News, 5, pp. 80-84.

Leong-Hong, B.W., & Plagman, B.K. (1982). Data dictionary/directory Systems. New York, NY: John Wiley & Sons.

Levine, A. (1985, April). Selecting a systems development methodology. Infosystems, 32, p. 72.

Martin, J., & Hershey, A. (1986, March). Software engineering depends on information engineering. Software News, 6, pp. 60-62.

Mazzucchelli, L. (1985, December 9). Structured analysis can streamline design. Computerworld, 19, pp. 77-86.

Michielsen, K. (1986, March 15). Micro applications development. Datamation, 32, pp. 96-98.

Miller, E. (1979). Tutorial: Automated tools for software engineering. Silver Spring, MD: IEEE Computer Society.

Myers, E. (1985, March 15). Getting a grip on tools. Datamation, 31, pp. 30-35.

Myers, W. (1978, February). The need for software engineering. Computer, 11, pp. 12-26.

Newport, J.P. (1986, April 28). A growing gap in software. Fortune, 113, pp. 132-142.

Oliver, P. (1985, Summer). Approaches to software engineering. Journal of Information Systems Management, 2, pp. 11-19.

Peters, L. (1981). A conceptual basis for software design standards. In Proceedings of the Software Engineering Standards Application Workshop (pp. 102-107). Silver Spring, MD: IEEE Computer Society Press.

Pfrenzinger, S. (1985, April 29). Too many tools spoil center. Computerworld, 19, p. 39+.

Poston, R.M. (1984a, July). Determining a complete set of software development standards: Is the cube the answer? IEEE Software, 1, pp. 87-89.

Poston, R.M. (1984b, April). IEEE software engineering standards. IEEE Software, 1, pp. 94-98.

Poston, R.M. (1984c, January). Software standards. IEEE Software, 1, pp. 95-97.

Pressman, R.S. (1982). Software engineering: A practitioner's approach. New York, NY: McGraw-Hill.

Proper methodology, micro link yield variety of benefits. (1984, August 20). Computerworld, 18, p. 17.

Ramamoorthy, C.V., Prakash, A., Tsai, W., & Usuda, Y. (1984, October). Software engineering: Problems and perspectives. Computer, 17, pp. 191-209.

Ross, D.T. (1976, November). Homiles for humble standards. Communications of the ACM, 19, pp. 595-600.

Ross, D.T., & Schoman, K.E. (1977, January). Structured analysis for requirements definition. IEEE Transactions on Software Engineering, 3, pp. 6-15.

Rush, G. (1985, October 7). A fast way to define system requirements. Computerworld, 19, pp. ID/11-16.

SECNAVINST 5000.1B. (1983, April 8). System acquisition.

SECNAVINST 5230.8. (1982, May 10). Information processing standards for computers (IPSC) program.

SECNAVINST 5231.1B. (1985, March 8). Life cycle management (LCM) policy and approval requirements for information system projects.

SECNAVINST 5233.1B (1979, January 25). Department of the Navy automated data systems documentation standards.

Sprague, L.R., Maibor, D.S., & Cooper, L. (1985, July 19). New DOD standard speeds software development. Government Computer News, 4, pp. 48-49.

Sprague, R.H., & Carlson, E.D. (1982). Effective decision support systems. Englewood Cliffs, NJ: Prentice-Hall.

Stevens, W.P., Myers, G.J., & Constantine, L.L. (1974, May). Structured design. IBM Systems Journal, 13, pp. 115-139.

Structured system analysis and its tools (Report AS40-050-501). (1985, July). Delran, NJ: Datapro Research Corporation.

Tausworthe, R.C. (1978). Standardized development of computer software: Part II, standards. Pasadena, CA: Jet Propulsion Laboratory, California Institute of Technology.

Thayer, R.H. (1984). The world of software engineering standards. In Proceedings of the Third Software Engineering Standards Application Workshop (pp. 149-156). Silver Spring, MD: IEEE Computer Society Press.

Thirteenth annual survey of performance related software packages. (1985). EDP Performance Review. Annual Reference Issue 1985, pp.1-63.

Twelfth annual survey of performance related software packages. (1984, December). EDP Performance Review, 12, pp. 1-39.

Wedburg, G.H. (1981). Implementation of software engineering standards. In Proceedings of the Software Engineering Standards Application Workshop (pp. 134-138). Silver Spring, MD: IEEE Computer Society Press.

Yourdon, E. (1979). Managing the structured techniques. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Yourdon, E. (1978). Structured walkthroughs. New York, NY: Yourdon Inc.

INITIAL DISTRIBUTION LIST

		No. 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.	LCDR Barry Frew, Code 54Fw Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	2
4.	Professor Kenneth J. Euske, Code 54Ee Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	1
5.	Ms. Jeanne L. Frew, Code 008 Fleet Numerical Oceanography Center Monterey, California 93940	2
6.	Curricular Officer, Code 37 Computer Technology Programs Naval Postgraduate School Monterey, California 93943-5000	1
7.	LT Margaret Lyle, USN 10213 Pumphrey Court Fairfax, Virginia 22032	1

END

2-87

DTIC