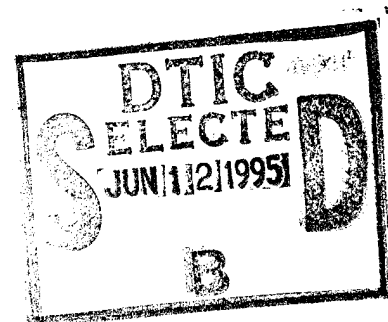


NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**CONFIGURATION MANAGEMENT FOR EXPERT
SYSTEM DEVELOPMENT: APPLICATION TO THE
MK 92 PROTOTYPE MAINTENANCE ADVISOR EXPERT
SYSTEM**

by

Paul Gregory Metzler

March 1995

Thesis Co-Advisors:

Magdi N. Kamel
Martin J. McCaffrey

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 3

19950608 053

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 1995		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE CONFIGURATION MANAGEMENT FOR EXPERT SYSTEM DEVELOPMENT: APPLICATION TO THE MK 92 PROTOTYPE MAINTENANCE ADVISOR EXPERT SYSTEM.			5. FUNDING NUMBERS	
6. AUTHOR(S) Metzler, Paul G.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum of 200 words) The Naval Postgraduate School in conjunction with Port Hueneme Division (PHD), Naval Surface Warfare Center is developing a diagnostic expert system for troubleshooting casualties in the MK 92 MOD 2 fire control system deployed in US Navy Oliver Hazard Perry class (FFG-7) guided missile frigates. The high turnover rate of student developers and the frequency with which changes are made to the expert system have highlighted a need for controlling the change process and the management of resources applied to implementing those changes to the expert system's knowledge base and software. This thesis develops a configuration management plan for the MK92 Maintenance Advisor Expert System (MK92 MAES) to assist project members in the management of changes to software and domain knowledge. The concept of configuration management is examined in detail with specific emphasis on the challenges of its implementation to expert systems. Two automated configuration management tools, CCC/Manager and PVCS Version Control, are evaluated for suitability for application in an expert system development environment. Finally, specific recommendations are presented for establishing a configuration management process for the MK92 MAES project.				
14. SUBJECT TERMS Expert Systems, Configuration Management, Diagnostic Expert System, Expert System Development Cycle, MK 92 MAES, MK92 Maintenance Advisor Expert System, Configuration Management Plan			15. NUMBER OF PAGES 206	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

Approved for public release; distribution is unlimited.

CONFIGURATION MANAGEMENT FOR EXPERT SYSTEM DEVELOPMENT:
APPLICATION TO THE MK 92 PROTOTYPE MAINTENANCE ADVISOR EXPERT
SYSTEM

by

Paul Gregory Metzler
Lieutenant, United States Navy
B.S., Boston University, 1988

Submitted in partial fulfillment
of the requirements for the degree of

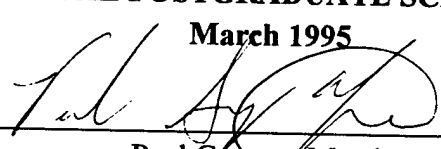
MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

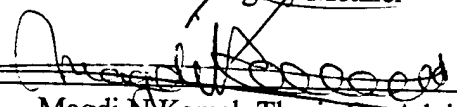
NAVAL POSTGRADUATE SCHOOL

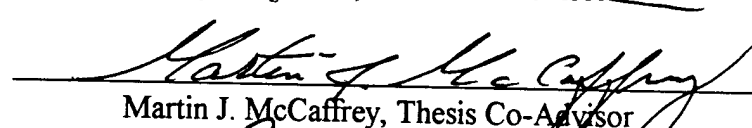
March 1995

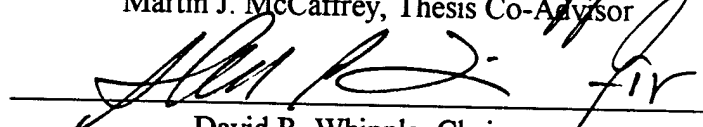
Author:


Paul Gregory Metzler

Approved by:


Magdi N. Kamel, Thesis Co-Advisor


Martin J. McCaffrey, Thesis Co-Advisor

 fir
David R. Whipple, Chairman
Department of Systems Management

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

ABSTRACT

The Naval Postgraduate School in conjunction with Port Hueneme Division (PHD), Naval Surface Warfare Center is developing a diagnostic expert system for troubleshooting casualties in the MK 92 MOD 2 fire control system deployed in US Navy Oliver Hazard Perry class (FFG-7) guided missile frigates. The high turnover rate of student developers and the frequency with which changes are made to the expert system have highlighted a need for controlling the change process and the management of resources applied to implementing those changes to the expert system's knowledge base and software.

This thesis develops a configuration management plan for the MK92 Maintenance Advisor Expert System (MK92 MAES) to assist project members in the management of changes to software and domain knowledge. The concept of configuration management is examined in detail with specific emphasis on the challenges of its implementation to expert systems development. Two automated configuration management tools, CCC/Manager and PVCS Version Control, are evaluated for suitability for application in an expert system development environment. Finally, specific recommendations are presented for establishing a configuration management process for the MK92 MAES project.

TABLE OF CONTENTS

I. INTRODUCTION	1
A. BACKGROUND	1
B. OBJECTIVES	1
C. RESEARCH QUESTIONS	2
1. Primary Research Question	2
2. Subsidiary Research Questions	2
D. SCOPE	2
E. METHODOLOGY	2
F. THESIS ORGANIZATION	3
II. A PROTOTYPE MAINTENANCE ADVISOR EXPERT SYSTEM FOR THE MK92 FIRE CONTROL SYSTEM	5
A. INTRODUCTION	5
B. COST BENEFIT ANALYSIS	6
1. Assumptions	7
2. Benefits	7
a. Reduced Repair Parts Costs	8
b. Manpower Savings	8
c. Improved Operational Readiness.	8
d. Reduced Reliance Upon Outside Technical Assistance	9
e. Improved Shipboard Training and Knowledge of MK92 FCS	9
3. Costs	10
a. Software Development and Associated Labor Costs	10
b. Deployment Hardware and Commercial Software Costs	10
c. Software and Hardware Maintenance Costs	11
d. Training Costs	12
4. Economic Analysis	12
a. Present Value Analysis	12
b. Savings/Investment Ratio	12
c. Discounted Payback Analysis	13

d. Sensitivity Analysis	13
C. EXPERT SYSTEM DEVELOPMENT CYCLE	14
1. Expert System Life Cycle Model	14
2. Problem Selection	15
3. Development Team	16
a. Domain Experts	16
b. Knowledge Engineers	17
c. Expert System Programmers	17
4. Knowledge Acquisition	17
a. Expertise of Expert	17
b. Technical Manuals	18
c. Other Recorded Sources of Knowledge	18
d. The Knowledge Acquisition Methodology	18
5. Knowledge Representation	21
6. Knowledge Coding	21
7. Knowledge Verification and Validation	24
D. EXPERT SYSTEM IMPLEMENTATION	25
1. Test and Evaluation	25
2. System Deployment	26
E. SUMMARY OF THE MK92 MAES PROJECT	27
III. CONFIGURATION MANAGEMENT CONCEPTS	29
A. OVERVIEW	29
1. Configuration Management Defined	29
2. Configuration Management and Change	31
3. The Purpose of Configuration Management	32
a. CM Establishes Software Integrity	32
b. CM Ensures Traceability	33
c. CM Increases Project Visibility	33
d. CM Supports Development and Maintenance	34
e. CM Improves Project Management	34
f. CM is an Integral Component of Process Improvement	34
4. The State of Configuration Management Practice	36

5. Configuration Management and the Product Assurance Disciplines	37
a. Quality Assurance	37
b. Verification and Validation	38
c. Configuration Management	38
d. Test and Evaluation	38
6. Sources of Configuration Management Guidance for DOD Programs	38
7. Configuration Management and the Software Life-cycle	39
a. Configuration Management's Role in Structured Design Methodology	39
b. Configuration Management's Role in Prototyping	40
B. CONFIGURATION MANAGEMENT TASKS	40
1. Configuration Identification of Software	42
a. Programs and Files	42
b. Software Identification Conventions	42
c. Configuration Identification of Files	43
d. Configuration Identification of Patch Files	45
e. Categorization of Software	45
f. Software Categorization in the MK 92 MAES Project	47
g. Configuration Item Naming Responsibility	47
h. Configuration Identification of Storage Media	48
i. Version Description Document	49
j. Configuration Identification of Documentation	50
2. Establishing Baselines	51
a. Functional Baseline	51
b. Allocated Baseline	51
c. Developmental Configuration	51
d. Product Baseline	52
3. Configuration Change Control	52
a. Configuration Change Control Activities	52
b. A Comment on the Configuration Change Control Process	53
c. Identify the Problem	53

d. Determine Appropriate Course of Action	55
e. Implement the Change	56
f. Configuration Control Board	56
g. Change Control Authority	58
h. Configuration Change Control of Specific Software Categories	59
i. Configuration Control Documents	62
j. Configuration Control Concerns	63
k. The Need for Automation in Configuration Change Control	64
4. Configuration Status Accounting	64
a. Overview of Configuration Status Accounting	64
b. The Configuration Status Accounting Process	65
5. Configuration Audits	66
a. Types of Configuration Management Audits	66
b. Functional Configuration Audits	67
c. Physical Configuration Audit	67
d. In-process Audits	67
e. Conducting Configuration Audits	68
C. THE CONFIGURATION MANAGEMENT LIBRARY	69
a. Working Libraries	69
b. Project Support Libraries	69
c. Master Library	70
d. Software Repository	70
e. Backup Library	70
1. The CM Library in Operation	70
a. Check-in Check-out Concept	71
b. Migration	71
c. Change Regression	71
d. Promotion	72
e. The Single User Approach to Using a CM Library	72
f. The Parallel Development Approach to Using a CM Library	72

D. CONFIGURATION MANAGEMENT KEY PERSONNEL	73
1. The Customer	74
2. Configuration Manager	74
3. Configuration Management Library Administrator	74
4. Individual Developers	74
5. Project Management	75
6. Configuration Control Board	75
E. THE CONFIGURATION MANAGEMENT PLAN	75
1. What is a Configuration Management Plan?	75
2. Developing CM Plans	76
3. Guidance on Configuration Management Plans	76
4. The Components of a CM Plan	78
5. Summary of Configuration Management	78
IV. ISSUES IN CONFIGURATION MANAGEMENT OF EXPERT SYSTEMS	79
A. GENERAL CM ISSUES OF EXPERT SYSTEMS	79
1. Lack of Detailed Specifications	80
2. Difficulty in Identifying Baselines	80
a. The Functional Baseline	80
b. The Allocated Baseline	81
c. The Developmental Configuration	81
d. The Product Baseline	82
3. CM Challenge Posed by Prototyping	82
4. Frequency and Sources of Change	83
5. Expert System Development Environment	83
B. CONFIGURATION MANAGEMENT OF THE KNOWLEDGE BASE	85
1. Characteristics of the Knowledge Base	85
a. Volatility	85
b. Expanding Functional Scope	86
c. System Size/Complexity	86
2. Distributed Knowledge	86
3. Knowledge Representation Scheme	87

4. Configuration Identification of Knowledge	87
a. Working knowledge	88
b. Captured Knowledge	88
c. Represented Knowledge	89
d. Product Knowledge	89
5. Baselining Knowledge	89
a. Functional Knowledge Baseline	90
b. Developmental Knowledge Configuration	90
c. Product Knowledge Baseline	90
6. Assessing the Impact of Changes to Knowledge	90
7. Controlling Changes to the Knowledge Base	91
a. What Influences the Change Control Process?	91
b. When Should Changes Be Made to the Knowledge Base?	92
c. To What Knowledge Category Should Change Control Extend?	92
C. CONFIGURATION MANAGEMENT OF EXPERT SYSTEM SOFTWARE IMPLEMENTATION	93
V. CONFIGURATION MANAGEMENT TOOLS	97
A. CRITERIA FOR SELECTING CONFIGURATION MANAGEMENT TOOLS	97
1. Configuration Management Tool Features	97
a. Change Control	98
b. Version Control	98
c. Reporting/Query Capability	98
d. Library/Repository	98
e. Release Management	99
f. Compatibility	99
g. Build Support	99
h. Team Support	99
i. Ability to Customize Features	100
2. Additional Selection Criteria	100
a. Cost	100
b. Ease of Incorporation Into a Project Life Cycle	100

c. Ease of Use	100
d. Security	101
e. Power	101
f. Robustness	101
g. Scalability	101
h. Quality of Commercial Support	102
i. Project Specific Features	102
3. Further Reading	103
B. AN EVALUATION METHODOLOGY FOR CM TOOLS	103
1. Conduct a Literature Review	103
2. Identify Constraints	103
3. Identify Evaluation Criteria	104
4. Identify Candidate Configuration Management Tools	104
5. Evaluate Candidate Configuration Management Tools	104
6. Make a Selection	104
C. APPLICATION OF EVALUATION METHODOLOGY TO THE MK92 MAES	105
1. Conduct the Literature Review	105
2. Identify Constraints	105
3. Identify Tool Evaluation Criteria	106
4. Identify Candidate CM Tools For Further Evaluation	107
5. CCC/Manager	107
a. Cost	107
b. Interface	107
c. Access Control	109
d. Life Cycle Modeling	111
e. Check-in/Check-out	112
f. Change Control	112
g. Version Control	112
h. Reporting Capabilities	113
i. Auditing	114
j. Interactive Merge	114
k. Compatible Operating Systems	115

l. Associated Products	115
m. Suitability to the MK 92 MAES Project	116
6. PVCS Version Manager	116
a. Cost	117
b. Interface	117
c. Access Control	118
d. Life Cycle Modeling	119
e. Check-in/Check-out	119
f. Change Control	119
g. Version Control	120
h. Reporting Capabilities	120
i. Version Labels	121
j. Keyword Embedding	121
k. Merge	121
l. Promotion and Migration	121
m. Compatible Operating Systems	122
n. Associated Products	122
o. Suitability to MK 92 Project	123
7. Evaluate Candidate Tools Using the Chosen Evaluation Criteria	124
8. Select a CM Tool	126
VI. CONFIGURATION MANAGEMENT IMPLEMENTATION	127
A. CM IMPLEMENTATION ISSUES	127
1. Organizational Issues	127
2. Skill level of Organizational Personnel	128
3. Organization's Level of Process Maturity	128
4. Technological Issues	128
5. Resources Available	129
B. PHASES OF CM ADOPTION	129
1. Phase 1: Determine CM Status and Needs	129
2. Phase 2: Evaluate Candidate CM Tools	129
3. Phase 3: Write the Configuration Management Plan	130
4. Phase 4: Implement a Pilot Project	130

5. Phase 5: Implement CM Plan	130
6. Phase 6: Evaluate and Adjust Plan as Necessary	130
C. CM ADOPTION PHASES OF THE MK92 MAES	131
1. Phase 1: The Status of CM and Needs of the MK92 MAES Project	131
a. Status of CM in the MK92 MAES Project	131
b. CM Needs of the MK92 MAES Project	132
2. Phase 2: Evaluate Candidate CM Tools for the MK92 MAES Project	133
3. Phase 3: Write the CM Plan for the MK92 MAES Development	133
4. Phase 4: Implement a Pilot Project	134
5. Phase 5: Implement the MK92 MAES CM Plan	134
6. Phase 6: Evaluate and Adjust Plan as Necessary	135
D. CM FOR THE MK 92 MAES	135
1. CM Organization	136
a. Project Manager	136
b. Student Project Leader	136
c. Configuration Manager	137
d. Configuration Library Administrator	137
e. Configuration Control Board	138
f. Programmers	138
g. NSWC-PHD and Domain Expert	139
h. Customers	139
2. CM Library	140
3. Configuration Identification for the MK 92 MAES Project	141
a. Baselines for the MK 92 MAES	141
b. Categorization of MK92 MAES Software	143
c. Configuration Identification Naming Conventions by Classification Level of Software	144
d. Configuration Identification Naming Conventions of Knowledge	145
e. Configuration Identification of Documentation Other than the Knowledge Document	146

f. Configuration Identification of Erasable Electronic Media	146
4. Configuration Change Control for the MK 92 MAES Project	147
a. Change Control Authority (CCA)	147
b. Change Control Process	148
c. MK 92 MAES Change Control Documents	148
d. Access Controls/Privileges	149
5. CM of the MK92 MAES Knowledge Base	150
a. CM Issues Related to MK92 MAES Knowledge Acquisition	150
b. The MK 92 MAES Approach to Implementing CM on the Knowledge Base	151
c. AllCLEAR	152
d. Changes to the Knowledge Base	154
6. Configuration Status Accounting recommendations for the MK92 MAES	155
a. Access Control List	155
b. Configuration Item Report	155
c. Change Status Report	156
d. PVCS Reports	156
e. Version Description Document (VDD)	157
7. Configuration Auditing Recommendations for the MK92 MAES	157
E. AN EXAMPLE OF THE CHANGE PROCESS	157
1. A Change to the MK92 FCS is Promulgated.	157
2. Implementing a Change to the Knowledge Base	158
F. CM RECOMMENDATIONS FOR AN EXPERT SYSTEM DEVELOPMENT CENTER	162
1. Assumptions	162
2. Recommendations for CM at an Expert System Development Center	163
VII. SUMMARY AND CONCLUSIONS	167
A. SUMMARY	167
1. How Can Configuration Management Concepts Be Applied To the Implementation Of An Expert System?	167

2. What are the Benefits of Implementing Configuration Management?	168
3. What Attributes of Expert Systems Present Unique Challenges, If Any, to Configuration Management?	168
4. What Issues Surround the Application of Configuration Management to Expert System Domain Knowledge?	169
5. How Can Automated Configuration Management Tools be Applied to a Configuration Management Program?	170
6. What Are the Implementation Issues Surrounding the Application of Configuration Management to the MK92 Maintenance Advisor Expert System?	170
B. RECOMMENDATIONS	171
1. Recommendations for Future MK92 MAES CM Initiatives	171
a. Network the MK92 MAES Project's Computers.	171
b. Identify CM Support Tools That Further Automate the MK92 MAES CM Process.	172
c. Send Prospective Configuration Library Administrators (CLA) to PVCS Training	172
d. Increase Secondary Storage for MK92 MAES Computers	172
e. Train NSWC-PHD Engineers to Use allCLEAR for Knowledge Representation.	172
2. Recommendations for Further Research	173
C. CONCLUSION	174
APPENDIX A. SOFTWARE CONFIGURATION MANAGEMENT STANDARDS	175
A. DEPARTMENT OF DEFENSE STANDARDS AND PUBLICATIONS	175
1. Department of Defense Standards	175
2. Military Standards	175
B. IEEE STANDARDS	175
C. INTERNATIONAL STANDARDS ORGANIZATIONS (ISO) STANDARDS	175
D. ELECTRONIC INDUSTRY ASSOCIATION (EIA) PUBLICATIONS	175
APPENDIX B. EXAMPLE OF A MK92 MAES CHANGE REQUEST FORM	177

APPENDIX C. PVCS REPORTS	179
A. EXAMPLE OF A PVCS ARCHIVE REPORT	179
B. EXAMPLE OF A PVCS DIFFERENCE REPORT	181
LIST OF REFERENCES	183
INITIAL DISTRIBUTION LIST	187

I. INTRODUCTION

A. BACKGROUND

The Naval Surface Warfare Center, Port Hueneme Division (NSWC-PHD) is seeking to improve the capability of shipboard technicians to determine, diagnose, and resolve problems occurring within the Mark 92 (MK92), Mod 2, Fire Control System (FCS). This highly complex system includes CAS/STIR radar, SM-1 Missile launcher and directors, Mark 76 gun mount, and various computers and interfaces that require continual maintenance. The cost of maintaining the system include, in addition to the cost of replacing a failed component, those associated with the reliance upon outside technical assistance in fault diagnosis and the replacement of circuit cards that have been mistakenly identified as failed components.

To reduce the costs associated with maintaining the system, NSWC-PHD and the Naval Postgraduate School are developing a knowledge based expert system to assist technicians onboard ships to diagnose and resolve problems occurring with the system. Modules covering the Calibration and Performance aspects of the system have been developed and are undergoing initial testing, verification, and validation. System implementation is primarily undertaken by Master's Thesis students. The high turnover rate of graduating students has underscored the need for the implementation of a process which will allow for continuity as development team members graduate, and new students are brought on board.

B. OBJECTIVES

This research is directed toward the development and application of software configuration management principles to the MK 92 Fire Control System Maintenance Advisor Expert System (MK92 MAES). The management of the change process is critical if a project is to allocate its resources efficiently. This is as true of expert system development as it is of traditional software engineering. To ensure the long term viability of expert systems, maintenance considerations must be at the forefront of development

considerations. Configuration management adds discipline to the change process of not only the expert system's code, but also the representation of the domain expert's knowledge.

C. RESEARCH QUESTIONS

This thesis attempts to answer the following research questions. The investigation of these questions shall form the basis for developing and implementing the configuration management program for the MK 92 MAES project.

1. Primary Research Question

- How can configuration management concepts be applied to the implementation of an expert system?

2. Subsidiary Research Questions

- What are the benefits of implementing configuration management?
- What attributes of expert systems present unique challenges, if any, to configuration management?
- What issues surround the application of configuration management to expert system domain knowledge?
- How can automated configuration management tools be applied to a configuration management program?
- What are the implementation issues surrounding the application of configuration management to the MK92 Maintenance Advisor Expert System?

D. SCOPE

The scope of the thesis is limited to the development of: (1) A configuration management approach to expert system development (2) An implementation of the configuration management plan to the MK92 MAES project.

E. METHODOLOGY

Research methodology for this thesis include a thorough literature review of configuration management theory and program implementation. In addition, interviews with software engineering professionals involved in the configuration management of software projects aimed at providing insight to the issues surrounding the establishment of

a configuration management process are conducted. Using the IEEE's suggested methodology for CASE tool evaluation, two CM tools are compared for the purpose of selecting the automated CM tool most appropriate for the MK92 MAES.

F. THESIS ORGANIZATION

This thesis consists of seven chapters and three appendices. The following is a brief description of the contents of each chapter.

Chapter II provides the reader with background information on the MK92 Maintenance Advisor Expert System. This background overviews the principles of expert system development.

Chapter III introduces the discipline of configuration management. In addition, the functional tasks of configuration management are examined in detail. A thorough review of the duties and responsibilities of key personnel involved in establishing and maintaining a configuration management program is also undertaken. Finally, the chapter discusses the contents and purpose of a configuration management plan.

In chapter IV, the discipline of configuration management is applied to the development of expert systems. Challenges presented by expert systems to the implementation of a CM process are discussed. Specific issues surrounding the maintenance of the knowledge base are also reviewed. Finally, an automated approach to managing the changes to an expert system's knowledge base is presented.

Chapter V examines the current state of the art of configuration management tools and the desired features for their application to a software design project. A comparison of two automated CM tools is made. The chapter concludes with the selection of the tool which is, based upon the characteristics of the project, best suited for application to the MK 92 MAES.

In Chapter VI, the principles outlined in Chapter III, and Chapter IV are applied to the MK92 MAES. Specific recommendations for the implementation of CM in the MK92 MAES project are made.

In Chapter VII, the thesis is summarized, and conclusions are drawn. Finally, recommendations are made for further research.

Appendix A is a listing of standards and guidance on configuration management and related topics, while Appendix B is an example of a Change Request Form for the MK92 MAES project. The last appendix, Appendix C, provides examples of a Change Difference Report and an Archive Report produced by the configuration management tool PVCS.

II. A PROTOTYPE MAINTENANCE ADVISOR EXPERT SYSTEM FOR THE MK92 FIRE CONTROL SYSTEM

A. INTRODUCTION

The 1990's have seen a steady trend of what has been referred to as the "downsizing" of the United States armed forces. Along with the resulting reduction in personnel and equipment has emerged an effort to identify technological alternatives that will reduce costs while enhancing operational readiness. The development of the MK92 Maintenance Advisor Expert System (MK92 MAES) is one such effort. It was undertaken by the faculty and graduate students of the Naval Postgraduate School in cooperation with system engineers of the Naval Surface Warfare Center, Port Hueneme Division (NSWC-PHD). The project is sponsored by the Naval Sea Systems Command (NAVSEA).

The MK 92 is the designation given to the fire control system (FCS) in operation on the United States Navy's Oliver Hazard Perry (FFG-7) class of guided missile frigates (FFG) and US Coast Guard medium and high endurance cutters class (WHEC 715-726). Additionally, the system has been deployed on the Australian Adelaide class frigates, Spanish Santa Maria class FFGs and Taiwan's Cheng Kuh class of guided missile frigate. (Jane's Fighting Ships, 1994).

As a fire control system, it is designed to coordinate the detection, tracking and engagement of hostile air and surface targets by the vessel's 76mm gun and missiles. The MK92 accomplishes this through the use of search/track radars, digital computers, servos, amplidyne, and other components, all largely reflective of 1970's technology. The FCS is modularized to support the maintenance concept of module replacement and the Navy's Planned Maintenance System (PMS), in an effort to minimize the number of personnel required to maintain it.

Maintenance of the MK 92 is conducted at the organizational (shipboard) and depot levels. At the shipboard level, Fire Controlmen (FCs) are limited to planned

maintenance, fault isolation, and corrective maintenance consisting of replacing modules, circuit cards, and minor Micro-miniature (2M) repair. (Lewis, 1993) If more extensive troubleshooting is required, technical representatives must be sent to the ship, no matter where it is, to isolate the problem and correct it. Equipment requiring repair outside the capabilities of a vessel's technicians are turned in to repair depots. All of this translates into increased system down time and higher maintenance costs.

During the period from July 1, 1989 to September 30, 1991, over forty percent of all initiated Casualty Reports (CASREPs) requested outside technical assistance in isolating the cause of the failure. Additionally, over twenty-two percent of all Depot Level Repairables (DLRs) turned in during fiscal year 1991 were found to be No Fault Evident (NFE), that is, in proper operating condition. In 1991, the associated cost was estimated to be \$700,000 per year. Faced with a decreasing budget and fewer technical representatives to send to ships, the Navy Surface Warfare Center, Port Hueneme Division (NSWC-PHD), recognized the need to improve the troubleshooting capability of the shipboard FCs. NSWC-PHD decided to investigate the possibility of using expert system technology as a possible remedy and approached the Naval Postgraduate School in the fall of 1992 for assistance. (Powell, 1993)

This chapter describes the development efforts of NPS faculty and students, and NSWC-PHD engineers. It is organized as follows. Section B introduces the cost benefit analysis applied to determine the feasibility of the MK 92 MAES project. Section C discusses the expert system development cycle of the MAES project. Section D takes a closer look at the implementation of the MK 92 MAES. Section E summarizes the project and provides some lessons learned.

B. COST BENEFIT ANALYSIS

Before commencing the development of the expert system, a cost benefit analysis was undertaken by a NPS graduate student to evaluate the feasibility of the effort. Using CASREP data, NFE information, and other sources of cost related variables, the officer was able to conduct a detailed analysis which looked not only at the costs and benefits as

compared to the status quo, but also the sensitivity of changes to various factors and their affect on the economic viability of the project. The following is a brief summary of the study's findings as determined by Powell (1993).

1. Assumptions

In order to conduct the cost-benefit analysis, the following set of assumptions was made:

- The MK92 MAES will be fielded to 39 ships.
- The program life of the system will be until 2005, the anticipated service life of the FFG-7 class.
- All quantifiable costs were multiplied by a 61 percent pertinency rate to account for the estimated percentage of casualties to the MK92 FCS in which the MK92 MAES would be useful.
- A 50 percent efficiency rate was applied to all cost savings to account for potential mistakes made by the expert system, as well as to make a more conservative estimate of MK92 MAES' impact. In other words all cost savings were reduced by 50%.
- Net present value calculations used a ten percent discount rate and discounted all cash flows to 1993 dollars.
- Personnel costs were computed by accelerating the composite wage rate by 32 percent to account for leave, medical care, and other fringe benefits.
- For purposes of calculating hardware costs, the useful service life of a notebook computer is assumed to be four years.
- The supply support response time resulting from decreased numbers of logistics related CASREPS (those in which the ship did not have the necessary part onboard) is assumed to be 5%.

2. Benefits

The study found the benefits resulting from the deployment of a maintenance advisor expert system include:

- reduced repair parts costs
- manpower savings
- reduced mean time to repair

- reduced reliance upon outside technical assistance
- improved shipboard training

a. Reduced Repair Parts Costs

The use of expert system technology would increase the likelihood of identifying the failed part correctly. Powell's analysis determined approximately 22 percent of all Depot Level Repairable parts (DLRs) are perfectly good parts. After applying the conservative efficiency rate of 50% to anticipated savings in terms of unnecessary parts expense, it was determined that \$215,748 per year could be saved in terms of parts alone.

b. Manpower Savings

Using expert system technology, onboard technicians would be able to troubleshoot faults much faster than relying on their own expertise and outdated technical manuals. Using the Fire Controlman Second Class (E5) Composite Rate, adjusting it for compensation and fringe benefits using a 32% acceleration rate, and reducing potential savings in terms of man-hours by 50 percent, an annual estimated savings of \$118,958 for 39 ships was determined.

c. Improved Operational Readiness

The analysis estimated MK92 FCS operational readiness would improve by 11%. This is due to a reduction in the mean time to repair casualties to the FCS and improvements in supply support response due to fewer NFE's and the associated burden they place upon the supply system.

(1) Reduced Mean Time to Repair. Since a Navy frigate's weapons systems need to be ready on a moment's notice, a one hundred percent system availability is always the goal. When a casualty does occur, repairs need to be prompt and effective.

The analysis determined the average trouble isolation intensive CASREP required 241 hours (over ten days) in maintenance downtime before the casualty was corrected. It was estimated that the use of the expert system would reduce the

downtime associated with casualties within the problem domain by 25 percent. This results in an eight percent increase in fleetwide MK92 MOD 2 FCS operational readiness.

(2) Improved Supply Support Response. Approximately 22 percent of DLRs are replaced unnecessarily, resulting in unnecessary parts expenses of over 10% for MK92 MOD 2 equipped ships. It is therefore conceivable that improvements to the supply system's response time could be achieved if the number of NFEs could be reduced. If a five percent improvement in supply related downtime could be achieved, the analysis determined over 4,030 hours of system downtime would be saved.

d. Reduced Reliance Upon Outside Technical Assistance

By using the MK92 MAES, a ship would reduce her reliance upon outside technical assistance, thus freeing up the time of the technical experts to focus their efforts on more critical casualties.

The MK 92 FCS program manager for Naval Sea Systems Command Pacific determined that 90 percent of all travel expenditures for fiscal year 1992 were for technical assistance travel. Of the travel expenditures for technical assistance, 85 percent were made in trouble isolation efforts. After accounting for those in which the MK92 MAES would be useful and applying the 50 percent factor to provide a conservative estimate, it was determined a savings of \$16,926 in travel expenditures by technical representatives could be realized.

e. Improved Shipboard Training and Knowledge of MK92 FCS

As FCs work with the MK92 MAES, it is foreseeable they will gain insight as to the thought process and approach an expert takes when troubleshooting a system. In addition, through use of the help features and explanation facility of the system, the FC begins to understand not only what to look for, but how to look for it. Technicians could then apply their increased troubleshooting skills to problems outside the domain of the expert system.

The system could also be used as a training tool to diagnose hypothetical casualties. In conjunction with circuit cards which have been intentionally faulted, the

MK92 MAES could be used to teach FCs, both on ship and in training schools, effective troubleshooting techniques.

3. Costs

The study estimated the following costs for the development of the MK92 MAES:

- software development and associated labor costs
- deployment hardware and commercial software costs
- maintenance costs
- training costs

a. Software Development and Associated Labor Costs

Like traditional software development, labor costs make up a majority of the costs associated with expert system development. However, most of the labor costs in an expert system, particularly when using a visual development environment, are associated with knowledge acquisition. Not only must the costs associated with the developers be included, but the cost of the expert's time must be included as well. Table 2.1 represents the estimated software development costs by fiscal year.

Fiscal Year	Estimated Software Development Costs
FY 1992	\$309,000
FY 1993	\$235,000
FY 1994	\$335,000

Table 2.1 Estimated Software Development Costs for the MK92 MAES.

b. Deployment Hardware and Commercial Software Costs

This category included the purchase and fielding of computers with the MK92 MAES. Table 2.2 lists the costs by computer type for hardware if fielded to 39 ships.

Additional runtime versions of the expert system shell are charged for under the licensing agreement of the tool in use by the MK92 MAES project, therefore only the initial development software need be included in the software costs. Software costs include the purchase of a database program which is to be used to incorporate parts data in the expert system. It is expected equipping 39 ships with a database program will cost approximately \$20,500.

c. Software and Hardware Maintenance Costs

Software maintenance costs include the cost required to report, identify, and implement any changes that may result from trouble reports received from the fleet. Additionally, the effect of ordinance alterations (ORDALTs) must be taken into consideration. It was determined that approximately 75 percent of a man-year would be required to properly maintain the system. This is estimated to cost approximately \$73,850 dollars per year.

In addition to the software maintenance costs, hardware will also need to be replaced or repaired. Hardware maintenance costs are estimated to be approximately \$12,000 annually.

System Features	Unit Price	Price for 39 Ships
386 Monochrome	\$1600	\$62400
486 Monochrome	\$1,900	\$74,100
486 Passive Matrix	\$2,600	\$101,400
486 Active Matrix	\$4,000	\$156,000

Table 2.2 Estimated Hardware Costs for the MK92 MAES from Powell (1993).

d. Training Costs

Although the MK92 MAES is designed to be user friendly, it was determined a one day introduction to the system's capabilities should be given to a ship's FCs at the time of deployment. Total training costs were estimated to be \$20,990.

4. Economic Analysis

Using present value analysis, savings/investment ratios, and discounted payback analysis, a quantitative approach to determining the economic feasibility of the MK92 MAES approach was accomplished.

a. Present Value Analysis

Present value analysis was accomplished by comparing the present value cost of developing, fielding and maintaining the MK92 MAES against that of the status quo. Table 2.3 represents the present value calculations with the Monochrome 486 monitor chosen for hardware implementation. The status quo's present value cost was determined to be \$7,868,422. The fielding of the MK92 MAES through 2005 was determined to have a Net Present Cost of \$6,822,201. This represented a potential net present savings of \$1,046,221.

Alternative	Net Present Value
Status Quo	\$7,868,422
MK92 MAES	<u>\$6,822,201</u>
Savings	\$1,046,221

Table 2.3 Present Value With Monochrome 486 from Powell (1993).

b. Savings/Investment Ratio

Savings/investment ratio (SIR) "is the relationship between future cost savings and the investment necessary to obtain those savings." (Haga & Lang, 1992) If

the SIR is equal to or less than one, the decision to make an investment should not be made on an economic basis alone. The SIR for implementation of the MK92 MAES varied from 3.033 using a 486 with a monochrome screen to 2.617 using a 486 with an active matrix color screen, indicating its implementation and deployment is a sound economic decision.

Computer	SIR
386 Monochrome	3.104
486 Monochrome	3.033
486 Passive Matrix	2.881
486 Active Matrix	2.617

Table 2.4 Summary of Savings/Investment Ratios by Computer Option from Powell (1993).

c. Discounted Payback Analysis

With discounted payback analysis, the shorter the payback, the more desirable the project. The discounted payback for the MK 92 MAES project was determined to be four years beyond fielding of the system.

d. Sensitivity Analysis

Due to the inherently uncertain application of economic analysis, a sensitivity analysis was conducted in an effort to anticipate the effect changes to policy and economic factors would have upon the feasibility of the project. The following factors were considered in the application of sensitivity analysis:

- Accelerated decommissioning. To identify how this would affect the feasibility of the project, the breakeven number of ships in commissioned service was calculated. Assuming all other variables remained constant, it was determined the MK92 MAES had to be deployed on twenty ships.

- Cumulative dollar value of all savings realized. It was determined, that even if program savings were overestimated by 45%, economic analysis would still be in favor of fielding the MK92 MAES.
- Sensitivity of repair parts savings. It was determined, all other variables remaining constant, that repair part savings could be reduced by 74% before breakeven would be reached.
- Trouble isolation man-hour savings. All other variables remaining constant, if no man hour savings were realized from deployment of the MK92 MAES, the system would still be preferable to the status quo.
- Technical representative travel savings. Regardless of the impact of the MK92 MAES on travel savings, its deployment would still be preferable to the status quo.
- Project delay. Powell (1993) determined, even if delayed by one year, the Navy would still receive \$755,063 dollars in discounted savings from MK92 MAES employment.

As a result of these findings, it was clearly evident that it would be prudent to proceed with the development of the expert system. Significant cost savings and operational readiness improvements would result.

C. EXPERT SYSTEM DEVELOPMENT CYCLE

This section discusses the expert system life cycle used to develop the MK92 MAES.

1. Expert System Life Cycle Model

The expert system life cycle (ESLC) model used by the MK92 MAES development team is a variation on the model presented by Prerau (1990). Prerau segments the ESLC into three phases; the initial phase, core development phase, and final development and deployment phase (Prerau, 1990).

The initial phase involves obtaining management approval, project team formation, domain selection, and hardware/software selection. These steps lay the foundation for the development process.

Core development includes an assessment of the project's feasibility and the implementation of a full prototype. Full prototype implementation includes knowledge acquisition, representation, and implementation.

Final development and deployment mark the final stage in Prerau's ESLC model. It is at this point the development team builds a final production system. The system is tested, evaluated, and known errors are corrected.

2. Problem Selection

As pointed out by Walters and Nielsen (1988), a broad problem domain can lead to ambiguity and a lack of direction for the development effort. As a result, designers of an application fall into the trap of designing a system which will attempt to do everything; the end result being an expert system which does nothing well.

To avoid this trap, it was important to define a bounded problem for the MK92 FCS domain to which the domain experts could construct a logical approach to its troubleshooting. NSWC-PHD engineers determined an expert system designed to diagnose problems associated with the Daily System Operability Test (DSOT) would be the best candidate for the initial effort.

The DSOT is a daily evaluation of a US Navy combatant's weapon systems, from the fire control radars to the weapons themselves. It provides a rapid and comprehensive means of assessing the availability of the ship's combat suite. In the process of conducting the DSOT, sailors inject simulated targets to evaluate the response of their fire control system and associated weapons against established standards. As a result, a hard copy summary of system functional performance is provided to the operator indicating any faults with the system.

Three primary areas are the focus of the DSOT. These are CAS/STIR transmitter RF Power Checks; DSOT initialization and calibration; and the performance test. The RF Power checks are conducted to ensure minimum required power is available to system components.

DSOT initialization and calibration, as the name alludes, is the phase in which the calibration of the MK92 FCS' fire control channels takes place. As each channel is tested in sequence, the system issues GO/NOGO status identifiers which are printed out each time DSOT is run. These GO/NOGOs flags are used by system maintainers as starting points in the troubleshooting process.

During the Performance Test, simulated targets are introduced into the system which the system attempts to detect, track, and engage. As with the DSOT Calibration test, a series of GO/NOGOs are printed out. Any time the system falls outside established parameters, a NOGO is issued. Similarly, the printout serves as a starting point for the FCs diagnosing the problem.

Because the GO/NOGO output format of the DSOT is the primary indicator of a system fault, it was an ideal candidate for selection as a domain boundary. Furthermore, as the DSOT output is the usual starting point for FCs troubleshooting the system, it would be a logical input to the proposed Maintenance Advisor Expert System.

3. Development Team

The following personnel made up the development team.

a. Domain Experts

Once the problem domain was identified, the next task became that of establishing the development team members. The engineers at NSWC-PHD were intimately familiar with the expertise of the various technical representatives for the MK92 FCS. As such, they were tasked with identifying the best candidates to be the domain experts. Their selection included one primary domain expert, Dorin Sauerbier, a technical representative under contract to the US Navy from UNISYS with almost 35 years of experience, and a secondary expert, Joe Guardione, an NSWC engineer with 18 years of MK92 experience. Both have extensive experience in all aspects of diagnosing casualties to the MK92 MOD 2 and were enthusiastic about contributing their time and knowledge in developing the expert system. In addition, Mike Roth was selected as a third domain expert to perform the validation.

b. Knowledge Engineers

The original intent was to train and use graduate officer students from the Naval Postgraduate school working on their thesis research as knowledge engineers during the knowledge acquisition phase. This proved to be unnecessary given the nature of the problem domain and the chosen knowledge representation scheme. As detailed in a later section, the domain expert was able to document his own knowledge directly without the assistance of an intermediary. In so doing, the traditional knowledge acquisition bottleneck created by an iterative interview process was substantially decreased. This was particularly important given the geographic separation of the developers and domain experts, and resulted in a smoother implementation effort.

c. Expert System Programmers

Though not involved directly in knowledge acquisition, the NPS graduate students were responsible, under the guidance of NPS faculty, for the design and implementation of the expert system in the selected development shell. In addition to coding, they conducted verification, validation, and testing. Additionally, they assessed the impact of changes to both schedule and budget, as well as other aspects of program management.

4. Knowledge Acquisition

The process of capturing domain knowledge was accomplished through several means.

a. Expertise of Expert

The primary source of knowledge was from the domain experts. With over 50 years of cumulative experience in fire control system diagnostics, mostly on the MK92 MOD 2 FCS, they were able to provide expertise and insight to troubleshooting which could not be obtained through examination of technical manuals alone. Their knowledge provided the heuristics on which much of the expert system was based.

b. Technical Manuals

Technical manuals supplemented the expert's knowledge, providing a resource to which they could refer to when documenting their expertise. The manuals, however, had to be used with caution. In some instances, they were inaccurate, requiring careful scrutiny by domain experts and other NSWC-PHD engineers. Fortunately, the extensive experience of the domain experts enabled them to recognize areas in which the technical manuals were inaccurate. In such cases, the engineers would consult other experts, or if necessary, consult manufacturers of specific components for additional information.

c. Other Recorded Sources of Knowledge

In addition to the domain expert's knowledge and technical manuals, information from other sources was used. One such source was Casualty Reports (CASREPs) requesting technical assistance. CASREPs include symptoms of a casualty, its cause, and corrective action taken.

Another source used was Ordinance Alterations (ORDALTs). ORDALTs are changes made to a weapons system such as the MK 92 FCS or a missile system. Included in these changes is a detailed documentation which has not been incorporated in the technical manuals. ORDALTs often provided a useful and more timely supplemental source of knowledge than the technical manuals.

d. The Knowledge Acquisition Methodology

Traditional knowledge acquisition techniques comprise an iterative process that consists of interviewing, eliciting the domain expert's knowledge, and testing that knowledge. This process requires close, repetitive interaction with the domain expert, creating a bottleneck in the development process. Additionally, such problems as interviewer bias, communication errors, and other anomalies can distort the expert's knowledge.

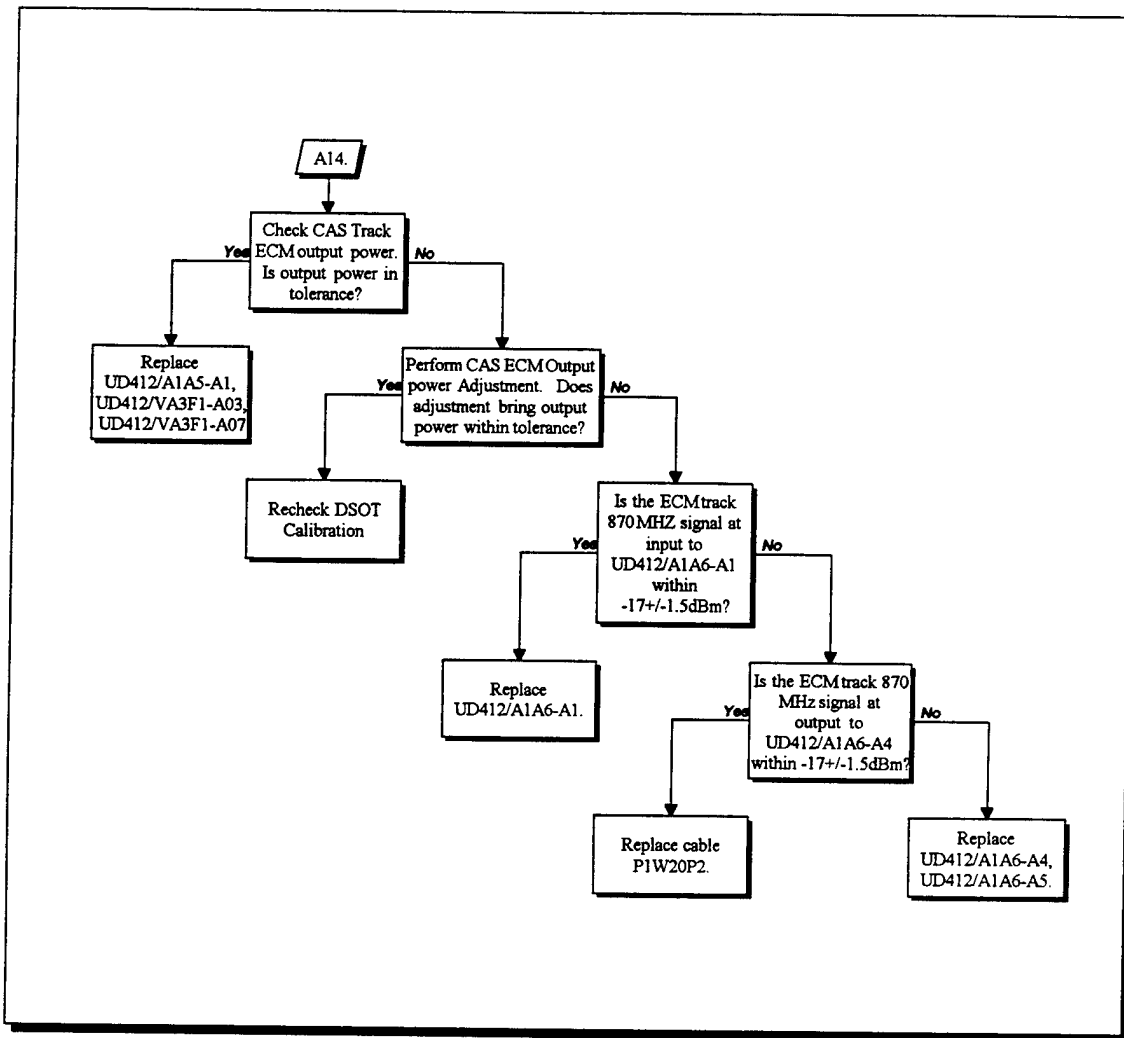


Figure 2-1. Example of a MK92 MAES diagnostic tree.

To enhance the efficiency of the knowledge acquisition process, the MK92 project team decided to have the domain experts develop graphical diagnostic trees which included their heuristics and troubleshooting procedures of the system's components. Diagnostic expert systems lend themselves well to this type of knowledge acquisition. Troubleshooting frequently requires following one or more paths of a hierarchical diagnostic tree. By using a diagnostic tree formalism, the experts were able to graphically represent their knowledge and review this knowledge for accuracy, thereby reducing the number of repetitive iterations in the knowledge acquisition process.

Figure 2-1 is an example of a diagnostic tree developed by the NSWC-PHD domain experts. As Figure 2-1 demonstrates, diagnostic trees represent a series of hierarchical questions which an expert would normally follow when diagnosing a problem. As the FC answers each question, they trace the thought process of the domain expert. Although yes/no questions are the most common type of questions asked, case statements could also be used to elicit one out of several responses.

The approach to knowledge acquisition involved developing a strategy for modularizing the problem domain. By breaking the problem area into modules, the domain expert could more easily concentrate on identifying the symptoms and troubleshooting procedures of a specific segment of the MK92 MOD 2 FCS.

The first level of abstraction was to segment the problem into calibration, performance, and RF Power check modules. These were separate subjects of the DSOT procedure with separate GO/NOGO output and parameters.

Once the main modules were identified, the next task was to divide each module into logical groupings in which similar symptoms, as evidenced by test output, would occur. To accomplish this, the domain experts began by identifying the instances of NOGO readings and grouping them according to potential cause.

First, a grouping was made according to which component the symptom (a NOGO) was identified. For instance, there are two primary radar components, CAS and STIR. Symptoms were identified according to whether they affected the CAS alone, STIR alone, or both.

Another level of abstraction is related to the mode of the radar in which the failure occurred. Modes include track mode and search mode, Electronic Counter-Measures (ECM), and others. If for example the symptom was a NOGO in the CAS portion of a test of the fire control system, but only in search mode of operation, a diagnostic tree would be created that would graphically depict the procedure for identifying the cause of the problem. This procedure of breaking the knowledge into

levels of different abstraction in a hierarchical structure enabled the domain experts to lay out and refine their troubleshooting strategy.

5. Knowledge Representation

In addition to knowledge acquisition methodology decisions, it was important to determine the method of knowledge representation. The challenge of knowledge representation lies in identifying a method which accurately depicts the expertise of the domain expert in such a way as to facilitate the knowledge coding process.

A rule-based paradigm was initially considered as the knowledge representation method of choice. It was noted, however, that as the system becomes larger, and the number of rules increases, a rule-based system's use and maintenance become exceedingly difficult. Additionally, as noted by Walters and Nielsen (1988), rule based structures are not well suited for representing procedural information.

A more flexible method of knowledge representation, the procedural network, was considered and ultimately selected as the knowledge representation method. Procedural networks, like flow charts, are graphical representations of the conditions which must exist before a conclusion can be reached. Each procedure is linked, defining the flow of logic within the network. Within each procedure is a series of instructions which are "executed," providing a vehicle for forward and backward chaining.

A main advantage of using procedural networks as a representation scheme for this application domain is its close match with the approach used by experts in diagnosing and resolving problems. In addition, procedural networks, are inherently modular. These two characteristics mirror the knowledge acquisition approach of the domain expert. As will be demonstrated in the following sections, the modularity and structure of procedural networks allowed for easy mapping from representation to implementation.

6. Knowledge Coding

The modular approach to building the knowledge base carried over to the knowledge coding process. As domain expert knowledge was acquired and knowledge

modules were completed, the implementation team began the task of mapping the knowledge as represented by the diagnostic trees to the expert system shell.

As with the representation scheme, the suitability of the selected tool to the type of knowledge being captured was a key consideration for the NPS developers. Prior to NPS' involvement in the project, unsuccessful attempts by NSWC-PHD engineers were made at implementing the acquired knowledge in an expert system shell that did not lend itself well to procedure-based knowledge representation. Several expert system shells were evaluated for their compatibility with the procedural knowledge of the MK92 MAES, and the expert system shell Adept, by Softsell, was selected (Lewis, 1993).

Adept is a visual expert system development tool which incorporates a graphical user interface (GUI) builder for the Microsoft Windows environment. Designed specifically for diagnostic expert system development, Adept implements knowledge as a collection of procedures, which are linked together to form a procedural network. (Smith, 1994)

Figure 2-2 is an example of the implementation of the knowledge obtained from the domain expert, shown in Figure 2-1. A comparison of Figure 2-1 with Figure 2-2 reveals the ease with which knowledge is mapped from the diagnostic trees, as represented by the experts, to the expert system shell. This close affinity between representation and implementation greatly enhanced the testing, validation, verification, and modification of the system.

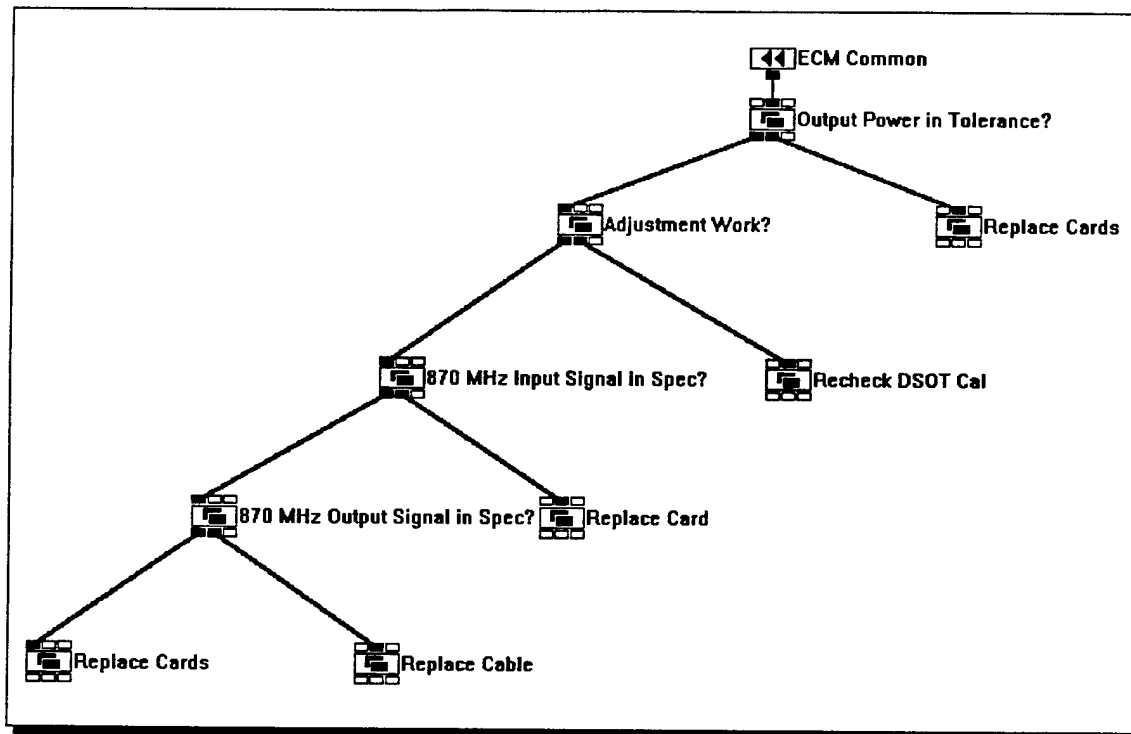


Figure 2-2. Knowledge coding using Adept procedures.

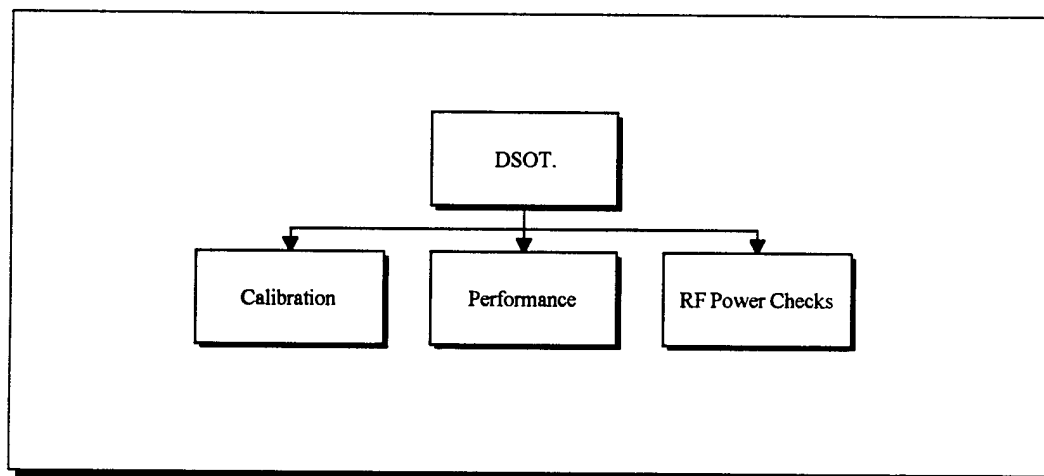


Figure 2-3. DSOT Modules.

To ease implementation of the domain expert's knowledge, the diagnostic trees were broken down into modules which represented the paths the domain expert would follow in his diagnosis (Smith, 1994). Each module was then implemented as a procedure in Adept. By maintaining correspondence wherever possible between the problem domain, diagnostic trees, and their implementation, the ability to perform maintenance and verification are enhanced. Figure 2-3 represents the three primary DSOT modules.

7. Knowledge Verification and Validation

During the MK92 MAES knowledge verification process, an effort was made to determine whether or not the domain expert's knowledge was accurate and complete. Independent verification of the knowledge was conducted by Mike Roth, an expert at NSWC-PHD with 20 years of experience in the MK92 FCS. The independent expert was responsible for evaluating the knowledge, identifying any discrepancies, and signing off on knowledge he considered accurate.

Once the knowledge was verified as accurate and complete, a validation process was initiated by the development team to ensure the implemented procedures accurately represented the knowledge document. Validation of the expert system was conducted by NPS graduate students and involved the comparison of the knowledge document with the corresponding expert system code. Completed expert system modules were then sent to the domain expert for further evaluation. Discrepancies that were identified by the domain expert were recorded on trouble reports and returned to NPS personnel for correction.

In both verification and validation, the ability to trace the knowledge as represented by the diagnostic trees to the actual code proved to play a key factor in establishing an effective V&V process. The Verification and validation team was able to quickly and easily verify and validate the operation of the MK92 MAES against the domain expert's knowledge. The logical representation of the domain experts knowledge and its ability to map easily to the Adept procedures, makes the V&V process faster, easier, and consequently cheaper than attempting V&V when traditional programming languages are used.

D. EXPERT SYSTEM IMPLEMENTATION

This section discusses the implementation of the MK92 MAES.

1. Test and Evaluation

Testing is the process in which the operation and behavior of the expert system is evaluated through the use of test cases (Dills & Tutt, 1994). Although the knowledge has been certified to be correct, and the implementation, was determined to be representative of the knowledge base, it was still necessary to determine whether or not the expert system functioned as intended.

Before the system could be fielded, a test and evaluation plan had to be established. To maximize the use of available resources, a test plan was developed which prioritized the order in which testing was to be completed. To accomplish this, MK92 MAES project team members examined CASREP information to identify high failure, high cost components which could be diagnosed by the expert system. Additionally, they attempted to maximize the number of diagnostic paths a particular test case would evaluate. (Dills & Tutt, 1994)

Working with fleet technicians at both Fleet Training Center Pacific (FLTRACENPAC) and the engineers at NSWC-PHD, NPS personnel established a testing order which would test the parts identified through CASREP analysis and path tracing in an order which was as convenient as possible for those conducting the evaluation. The test cases were implemented by both FLTRACENPAC and personnel operating a shore based mock-up of the MK 92 MOD 2 FCS at NSWC-PHD. (Dills & Tutt, 1994)

Test results were provided on verification sheets, denoting any problems encountered. Those identified were first evaluated by the knowledge coders, to ensure an error in implementation was not made, with the remaining passed to the domain expert for evaluation.

2. System Deployment

Before the system was deployed, considerable effort was expended to demonstrate the system to fleet sailors. Visits to ships on the waterfront, presentations at navy-wide technology expositions, and demonstrations at Navy training commands were used to gauge enthusiasm for the product as well as gain further insight as to deployment issues. Questions regarding the type of computer to be deployed (desktop vs. laptop), what commands to receive evaluation copies, and suggestions for enhancements were sought.

To demonstrate the capability of the system in an effort to sustain management support for the project, it was determined the system would be deployed in phases. The first phase would consist of the first two modules; calibration and performance. Power checks would be developed and fielded based upon the success of the first two. In so doing, the system could serve as a proof of concept while at the same time minimizing economic risk by initiating further development.

The decision was made, after an evaluation of alternatives, to provide a copy of the expert system to a frigate preparing to deploy, the USS Sides (FFG-14). Another evaluation copy was provided to the Navy's MK92 FCS school. Briefings were conducted with all levels of shipboard management, from the commanding officer to the actual technicians who would be using the system. The system was deployed on a commercial off the shelf (COTS) notebook computer which allows the FC to take the expert system to the location of the casualty for troubleshooting.

Initial feedback has been very positive. Upon returning from her deployment, USS Sides sent a message to NSWC NSWC-PHD, with copies to the Commander in Chief, Pacific Fleet (CINCPACFLT), the Naval Surface Warfare Center (NSWC), and Commander, Naval Sea Systems Command in which the USS Sides stated its evaluation of the MK92 MAES. The USS Sides (1995) noted, "MAES correctly diagnosed and recommended the proper corrective action for all faults" which were within the domain of the expert system. By their estimates, during a three month deployment, the MK92 MAES saved 30 man hours in troubleshooting and provided over 40 man-hours of training. Their

confidence in the MK92 MAES led them to recommend further testing of the system be carried out and implementation be considered for all MK92 MOD 2 frigates (USS Sides, 1995).

In addition to its deployment in USS Sides (FFG-14), the MK92 MAES was recently used to troubleshoot a casualty onboard USS John A. Moore (FFG-19). USS John A. Moore was receiving a technical assistance visit by NSWC engineers during a port visit to NSWC Port Hueneme, CA. The ship was experiencing a casualty which was resulting in NOGOs for all readings in the calibration portion of the DSOT. Ship's force had been troubleshooting the casualty for approximately one week (Torres, et. al., 1995). Using the MK92 MAES, NSWC engineers and USS John A. Moore FCs were able to successfully isolate the problem in approximately 15 minutes (Seto, 1995).

E. SUMMARY OF THE MK92 MAES PROJECT

The MK92 MAES represents a proof of concept in diagnostic expert systems and their role in the US Navy. With the potential for significant savings in terms of dollars, and time, the MAES promises to provide sailors with a diagnostic tool which can alleviate their need to rely upon outside assistance. The result is an improvement in combat readiness, reduced repair parts costs, manpower savings, reduced dependence upon outside technical assistance, and enhanced training .

The use of diagnostic trees and a visual expert system development environment eased the processes of knowledge acquisition, representation, and coding. The procedural structure and intuitive representation in both the diagnostic trees and expert system shell represent an instance where due consideration was given to matching the knowledge representation and implementation to the problem domain. The result was a modular system which provided for easier maintenance and evolution.

Though further testing and evaluation has yet to be done, all indications are that expert systems have matured to the point they can play a role in the everyday activities of the sailor. As budgets decrease, and personnel depart, the need to capture the knowledge

of the Navy's "experts" increases. The MK 92 MAES represents one such effort to move expert systems out of universities and laboratories and onto the "front lines."

III. CONFIGURATION MANAGEMENT CONCEPTS

This chapter explores the principles of configuration management. Several definitions are presented. First, the relationship of configuration management to other disciplines associated with software engineering is discussed. Next, the functional tasks of the configuration management process are examined in detail. In addition, configuration management's role in the software development life cycle is explored. Finally, the reader will be introduced to the concept of the configuration management plan.

A. OVERVIEW

Software configuration management can trace its ancestry to DOD and other government agency's projects in the 1960's (Berlack, 1992) (Buckley, 1993). As the complexity of systems increased, engineers and managers alike began searching for a methodology to control the development process. Out of their early efforts grew the discipline of configuration management.

Though originally applied to hardware, configuration management principles have been adapted and adopted by developers of software, documentation, drawings, and multi-media (Berlack, 1992) (Buckley, 1993) (Tomayko, 1990). Configuration management has enabled project managers to establish improved methods for managing change. The following subsections expound upon the concept of change management. For further information on standards, and recommended practices in configuration management, the reader is referred to Appendix A.

1. Configuration Management Defined

There are many definitions of configuration management. However, within each is a recurring theme: the management of change. The life cycle of software, from specification until its removal from service, is evolutionary. As the end-user's needs change, bugs are detected, or enhancements are introduced, some process is established which takes these changes, controls them, and turns them into a product. This process has

come to be known as configuration management. The degree to which the intended change is reflected in the actual change is the barometer of the quality of the CM process.

Several definitions of configuration management have been introduced by the Institute of Electrical and Electronics Engineers (IEEE), Department of Defense (DOD), and authors of various publications on the subject (Bersoff, 1984) (Buckley, 1993) (Berlack, 1992) (Tomayko, 1990). Their interpretations are presented below.

The Software Engineering Institute (SEI) definition, as promulgated in their curriculum module on software configuration management states:

Configuration management encompasses the disciplines and techniques of initiating evaluating and controlling change to software products during and after the development process. (Tomayko, 1990)

Bersoff (1984), a senior member of the IEEE, further defines CM as:

the discipline of identifying the configuration of a system at discrete points in time for the purpose of systematically controlling changes to the configuration and maintaining the integrity and traceability of the configuration throughout the system life cycle.

The definition used by both Buckley (1993) and Berlack (1992) is derived from MILSTD 973B, and reads as follows:

Configuration management is a discipline applying technical and administrative direction and surveillance to:

- identify and document the functional and physical characteristics of configuration items (CI)
- audit the configuration items to verify conformance to specifications, interface control documents, and other contract requirements
- control changes to configuration items and their related documentation
- record and report information needed to manage configuration items effectively, including the status of proposed changes and the implementation status of approved changes (Buckley, 1993) (Berlack, 1992)

2. Configuration Management and Change

Each of these definitions, slightly different in semantics, shares a theme which will be reiterated throughout this thesis: configuration management is the process of controlling change. Since the interpretation of the word change may be different from one reader to another, it is important to distinguish the author's intent as to how the word "change" should be interpreted. The term "change," when employed in this thesis, should be thought of as a generic descriptor for the types defined below.

This thesis utilizes the Software Engineering Institute's (SEI) explanation of software change types. (Tomayko, 1990)

SEI separates change into two major categories. The first category is discrepancies. Discrepancies are broken down into three major types:

- Requirements errors. Requirements errors, as the term suggests, result from an error in establishing the user's requirements. Tomayko (1990) states, "either the customer or marketing did not fully or clearly express the requirements, or incorrect information was given."
- Development errors. Development errors result from the incorrect implementation of correct requirements. In other words, the requirements correctly stated what the software should be able to do, but the software cannot accomplish the task correctly.
- Violations of standards. This third type of discrepancy refers to a violation of agreed upon development standards. Generally, the standards which were to have been adhered to were stated in the contract, but were not correctly implemented during development.

The SEI's second category of change is termed requested changes. Typically, there are three kinds of requested changes:

- Enhancements. Enhancements are change requests that involve the establishment of new requirements. These include the addition of greater functionality to the product. An example of an enhancement would be the addition of reporting features to a software product.
- Improvements. SEI considers product improvements to be changes to software that do not involve functionality or performance. One example might be the changing of text color to make it more readable. Another might be to

improve system documentation to make it easier to understand. (Tomayko, 1990)

- Unimplementable requirements. A requirement might not be implementable if resource constraints prohibit development of software that encompasses its intended operation. This would be a lack of funding necessary for the implementation of an established requirement. In such a case, the customer and developer may agree to change the software to adhere to a level of functionality that can be accomplished within the budgetary constraints. (Tomayko, 1990)

3. The Purpose of Configuration Management

Why should an organization undertake the daunting task of establishing a configuration management program? After all, on the surface, it would appear as though the added layers of review and administration would only serve to take up time. This, however, is a superficial view of configuration management. It does not take into account the benefits CM can have, not only on project management, but also in development of the final product. The following paragraphs provide a more in depth look at the potential benefits of configuration management.

The definitions presented in the previous subsection allude to the potential benefits of CM. Terms such as traceability, integrity, and control are key words in the goals of any project manager. CM supports software product integrity, increases traceability, and provides a vehicle for controlling the development process. Additionally, CM can act as a medium for increasing the visibility of a project. It is these potential benefits which have prompted organizations to adopt CM as one tool in their bag of tricks for productively developing and maintaining quality code.

a. CM Establishes Software Integrity

The goal of the software developer, according to Bersoff (1984), is to develop a product which matches, as closely as possible, the needs of the customer. This is known as product integrity. Applying his definition of product integrity (Bersoff, 1984) to software, the following definition of software integrity emerges:

Software integrity describes a set of characteristics of a software product that:

- satisfies the customer's functional needs
- has traceability throughout the entire software life cycle
- fulfills specified performance requirements
- is delivered and maintained within cost and delivery constraints

CM ensures software product integrity, version control, and change control. Mechanisms in the CM process prevent untested modules from being included in release copies of a product. Likewise, change control implies a procedure for the identification, review, impact assessment, and implementation of changes to a configuration item.

b. CM Ensures Traceability

Traceability refers to the ability to identify each configuration item, regardless of the phase of a product's life cycle, to the original specifications. Traceability must be maintained throughout the software development life cycle (SDLC). From project inception, starting with requirements definition through evolution of the product, until it is discarded, CM can ensure the traceability of a configuration item.

c. CM Increases Project Visibility

As software maintenance costs skyrocket, and budgets decrease, greater emphasis is being placed upon software development. This is as true in the corporate sector as it is in DOD. CM, through the reporting and auditing processes of status accounting and configuration audits, acts as a vehicle for communication of the status of a software product and its associated documentation. Increased visibility allows developers and project managers the opportunity to adjust their resources before they discover, all too late, a project is over budget and behind schedule. (Berlack, 1992)

d. CM Supports Development and Maintenance

CM supports development and maintenance through version control, change control, documentation control, and CM development tools. Version control enables the developer to recall any previous version of a configuration item. This can be invaluable when attempting to identify the source of a software problem. (Buckley, 1993)

Change control provides a process for the identification, impact analysis, and implementation of changes to software, regardless of scope or origin. CM is the management of change, and plays the primary role in effective change control.

Configuration management also ensures user manuals, code descriptions, and other items are not lost, or inconsistent with other aspects of the project. Through status accounting and configuration audits, inconsistencies are identified for correction.

Today, many of these features are incorporated in automated CM tools, thereby simplifying the tasks of both developer and configuration manager. A more detailed review of CM tools, their capabilities, and limitations, is undertaken in Chapter V.

e. CM Improves Project Management

Today's business environment has forced product managers to take the same perspective of software development efforts as a line manager would for a production line. This is driven by management's increasing recognition that software is a valuable company resource. As software development becomes a key component to an organization's competitiveness, focus is on product delays, and cost over runs. CM is one tool management can use in their efforts to achieve higher productivity and greater cost control.

f. CM is an Integral Component of Process Improvement

The business climate of the '90s can be characterized by an emphasis on process improvement (Curtis, 1992). "Total quality" is the buzzword of the day in both corporate America and DOD.

This trend has extended to software development. One version of process improvement gaining acceptance in the software engineering community is the Capability Maturity Model (CMM). As the name implies, the CMM attempts to classify the degree to which an organization's software engineering process has evolved. (Curtis, 1992)

The CMM's fundamental premise is a recognition that technology and dollars alone cannot be relied upon to ensure a software product has an acceptable level of quality (Curtis, 1992). An organization must adopt processes that create an environment in which quality software is a by-product of day to day operations.

The CMM is divided into five levels, with the fifth level being the highest level of maturity. The following is a brief description of the CMM's levels and the key process areas associated with each (Curtis, 1992).

- Level 5 Optimizing: Focus is on continuous process improvement. Key process areas include: defect prevention, technology innovation, and process change management.
- Level 4 Managed: Focus is on product and process quality. Key process areas include: process measurement, and analysis; quality management.
- Level 3 Defined: Focus is on engineering process. Key process areas include: organization process focus, organization process definition, peer reviews, training programs, inter-group coordination, software product engineering, and integrated software management.
- Level 2 Repeatable: Focus is on project management. Key process areas include: software project planning, software project tracking, software subcontract management, software quality assurance, software configuration management, and requirements management.
- Level 1 Initial: Focus is on heroes, meaning star programmers. No key process areas are in place.

To achieve level 2 of the CMM, a sound configuration management plan must be in place. This underscores, the degree to which CM is fundamental to a mature software engineering process.

4. The State of Configuration Management Practice

The extent to which configuration management is being practiced by software developers is varied. Capers Jones (1994) states as of 1993:

- 30% of US software producers have no configuration control automation.
- 30% have source code automation, but no documentation automation.
- Almost 30% have both source code and documentation automation, but are not integrated.
- Only 10% were moving toward fully integrated configuration control. No differentiation between the degree to which CM was an integral part of the SDLC was offered.

In 1990, a delphi study regarding software maintenance problems (Deklava, 1992) was conducted using attendees of the 1990 Annual Conference on Software Maintenance Association as participants. From that study, "system documentation incomplete or nonexistent" tied with "performance measurement difficulties" for third place of the top twenty major problems in software maintenance. Two respondents pointed out the expense of training new maintainers and lowered maintenance productivity as a result of poor software documentation (Deklava, 1992). The respondents identified two potential benefits of implementing a configuration management plan; cost control and improved software maintenance productivity.

Not all respondents were in agreement as to the extent of the problem of poor system documentation. Two participants stated their personnel had a lot of expertise with their system, and therefore did not rely upon the documentation. Three respondents, who dismissed the need for documentation, stated their preference for source code due to the fact the documentation was "not reliable or specific enough." (Deklava, 1992)

Those stating they did not need documentation for their system, ironically, pointed out two reasons one should have a configuration management program in place. What if an organizations "experts" suddenly left, or a change was made to the systems which was unfamiliar to them? A lack of proper documentation could be detrimental to the maintenance of the system.

Why is system documentation so poor that one is forced to rely upon source code? Configuration management, when properly adopted by an organization, becomes a cornerstone to a software design process which enhances the production of accurate and reliable documentation as a natural by-product of coding.

Another finding was the level of CM knowledge possessed by personnel in software development organizations. One-thousand-four-hundred software developers in North America participated in a standardized test on CM conducted by R.S. Pressman and Associates. Out of a maximum score of 100%, managers scored 60% and developers scored 50% in their level of configuration management knowledge (Computerworld, 1994). The Computerworld article did not address how many of the 1400 surveyed had a configuration management plan implemented within their organization.

The validity and consistency of the instruments used in these surveys was not established. However, if they are even remotely accurate measures, both developers and managers have a lot to learn about configuration management. Admittedly, one cannot use three surveys to gauge with precision, the state of CM in software production organizations. The findings, nonetheless, serve as indicators of a lack of knowledge and/or acceptance of CM's role in the development process.

5. Configuration Management and the Product Assurance Disciplines

"Configuration management is an integral part of software development across all phases of the life cycle" (Tomayko, 1990). As such, it interacts with other software engineering disciplines.

Bersoff (1984) identifies configuration management as belonging to a set of disciplines he describes as "product assurance disciplines." These include:

a. Quality Assurance

Quality assurance (QA) when referenced to software development, involves techniques which developers apply to insure the software "meets or exceeds pre-specified standards during a product's development life cycle" (Bersoff, 1984). In the

absence of specific quality requirements, QA uses "industrial or commercially acceptable levels of excellence" (Bersoff, 1984).

b. Verification and Validation

Verification and Validation (V&V) is a discipline which has two aspects. The first task, verification, determines whether the product adheres to established requirements, such as functional specifications in a contract. (Bersoff, 1984)

Validation on the other hand, goes one step further. Validation tests whether or not the software not only meets agreed upon requirements, but does what the end user intended (Bersoff, 1984). For further information on verification and validation, and its application to expert system development, the reader is referred to Dills & Tutt (1994).

c. Configuration Management

Configuration management acts as a product assurance discipline by acting as the repository for all documentation and code related to the project, maintaining a history of development, storing and controlling product versions, and evaluating and controlling changes. As a central controlling and storage facility, CM is integral to developing quality software. (Bersoff, 1984)

d. Test and Evaluation

Test and evaluation (T&E) refer to a process in which and organization, independent of the development group, evaluates the software for correctness and conformance to stated objectives (Bersoff, 1984).

6. Sources of Configuration Management Guidance for DOD Programs

DOD has recognized the benefits of configuration management and as a result, has published requirements for the establishment of CM programs by organizations developing software for DOD. Additional guidelines are found in the publications and standards of international bodies such as IEEE and the International Standards Organization (ISO).

Those interested in establishing a software configuration management program are referred to Appendix A for further information.

7. Configuration Management and the Software Life-cycle

The implementation of configuration management should begin as early as practicable in a software development project. All too often, configuration management is not considered until there is a problem. Only when project management discovers changes are undocumented, the program is over budget, or behind schedule, do they look to CM as one solution to their problems. (Berlack, 1992)

Requirements for CM should be established when drafting the agreement between the customer and developer. In this way, CM can become an integral part of the software development life cycle (SDLC). CM has a role in the entire SDLC, from the first requirement established, until the software is removed from service (Berlack, 1992) (Buckley, 1993).

a. Configuration Management's Role in Structured Design

Methodology

Configuration management's role in structured design methodologies is fairly well established. Though a discussion of the SDLC is beyond the scope of this thesis, a cursory review of its phases is provided in Dills & Tutt (1994). More complete discussions are found in Smith (1994) and Berlack (1992). A brief listing of the phases of a traditional SDLC is provided below:

- Concept definition
- Requirements definition
- Design
- Implementation
- Integration and Testing
- Operations and Maintenance

CM supports each of these phases by capturing all relevant information regarding a design in such a way that it can be recalled for use at any time. In addition to software versions, requirements, test plans, documentation, and changes are also recorded. (Buckley, 1993) (Berlack, 1992) (Tomayko, 1990)

b. Configuration Management's Role in Prototyping

As Hull and Kay (1991) point out, prototypes at some point in their SDLC transition to more traditional cycles of software development. The establishment of a CM process early in the prototyping stages of a development effort can make this transition a smooth one. CM captures the change history and other relevant files and documentation associated with the software's development.

Often the approach to designing a prototype does not take configuration management into account. Developers have the attitude, "we don't need CM, we're only going to throw this away anyway." Prototypes have a way, however, of evolving into the actual deliverable. Traditional thought on prototyping revolves around the evolution of interfaces, and programming techniques. However, as Kolkhorst (1994) points out, one can also "prototype" the procedures and policies of configuration management.

Regardless of what development methodology is used, configuration management is critical to managing the changes common in any software's life cycle. CM should evolve with the software development process, adjusting as needed, yet always capturing and managing change. (Berlack, 1992) (Buckley, 1993) (Tomayko, 1990)

B. CONFIGURATION MANAGEMENT TASKS

This section examines the primary elements of configuration management in greater detail. The four major tasks of configuration management; identification, change control, status accounting, and auditing are described. Typical methods for their application are also discussed. The applicability of these techniques will vary from project to project. The key to successful implementation of a CM process is to scale it to the project's specific needs. Figure 3-1 depicts the configuration management tasks.

The reader is cautioned that not all aspects of CM discussed in the following sub-sections may apply to their project. They are intended to serve as guidelines. The use of a particular CM tool will often drive the method of implementation, particularly with regard to configuration identification and change control. The remaining chapters of this thesis provide an example of the application of CM to the MK92 MAES project.

Configuration management is applied to all aspects of a project. In addition to the software code, CM controls (Tomayko, 1990):

- requirements
- specifications
- design documents
- source code
- object code
- test plans
- test suites
- maintenance manuals
- user manuals
- interface control documents
- memory maps

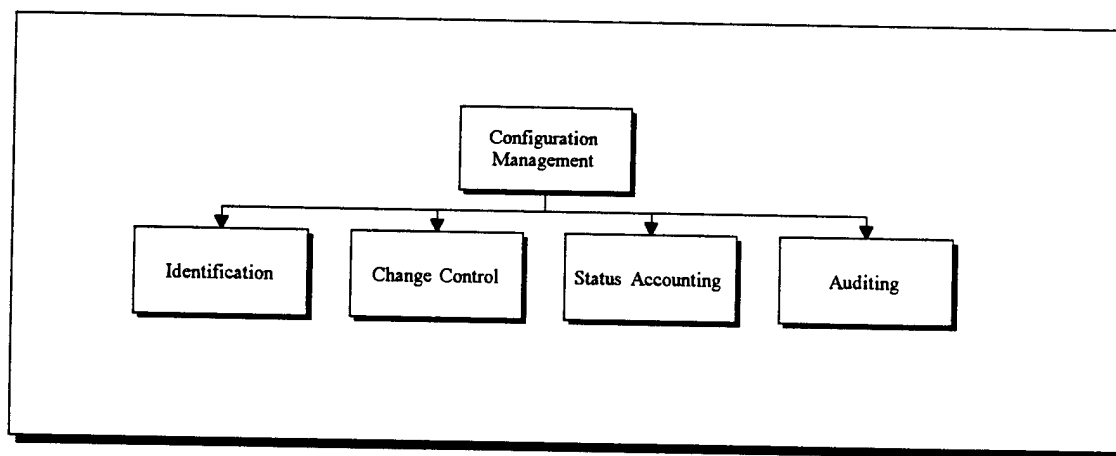


Figure 3-1. Configuration management tasks.

Although hardware and other components listed above are often under configuration management, in the interest of limiting the scope of this thesis, only issues related to the configuration management of software and the associated documentation will be explored. Therefore, the term configuration management shall refer specifically to software configuration management unless otherwise noted.

The only exception to this policy is the CM of storage media. The use of storage media such as floppy diskettes enables software to be distributed for use at sites other than where it is developed. Because of the close relationship between software and its method of storage, the CM of storage media is addressed.

1. Configuration Identification of Software

Configuration identification (CI) involves the recognition of what software items are key to maintaining the integrity of a project under development, and placing them under configuration control. (Buckley, 1993) (Berlack, 1992) This subsection examines the conventions for the configuration identification of a generic software program. Chapter VI offers a recommended approach for incorporating CI into the MK 92 MAES configuration management process.

a. Programs and Files

Before software configuration identification can be further examined, it is important, to ensure the reader understands the distinction between the terms "file" and "program." The term "file" is refers to a physical and logical collection of software code .

The term "computer program" or "program" refers to a collection of files. (Buckley, 1993)

b. Software Identification Conventions

When applying CM to hardware, one finds it can be identified at innumerable levels of abstraction. Software, however, is typically identified at two: the configuration item, known as computer program level and file level. (Buckley, 1993)

In DOD, at the program level, each item has what is referred to as a computer software configuration item (CSCI) identification number. A CSCI is a labeling scheme for software configuration items. The CSCI number provides a common reference for those items that make up the computer program, and serves as an "address for all actions and documentation related to that computer program." (Buckley, 1993)

Information one might wish to contain within an identifier at the program level includes the program's name, an abbreviation or acronym for the program's name, and an ID number. (Buckley, 1993)

Software code is identified and controlled at the file level. In general, there are three types of software files to be identified. These are source files, object files, and executable files. (Buckley, 1993) (Berlack, 1992) (Tomayko, 1990)

Again, DOD has its own terminology for a file. DOD STD-1267A refers to files as "controlling units" (DOD, 1988). This naming convention was intended to avoid any language-specific terminology, such as procedure, module, or function. However, as argued by Buckley (1993), the term "controlling unit" is also ambiguous, and therefore, are not used in the context of this thesis. Instead, the terms "file", or "configuration item," are used in recognition of the method in which most programs are represented when stored on disk. (Buckley, 1993)

c. Configuration Identification of Files

Why is it necessary to further discriminate between files? To attempt control of software configuration items at the program level would be impossible. Computer programs in large systems often contain tens of thousands of lines of code bundled within dozens of files. Considering the need for changes to various files; some more often than others, it is unfeasible to issue revisions for the entire program after each change is instituted. A more logical, approach is to apply control at the file level, and only issue those files which were actually modified. (Buckley, 1993)

Software at the file level, is composed of three generally accepted identifying elements (Buckley, 1993):

- file name
- file type
- version number.

Buckley uses the following symbology to identify the file: the underscore, period, and semi-colon are part of the identification scheme, and are typically included in each instance of a file identifier. An example is presented for clarification. (Buckley, 1993)

PROGRAM ACRONYM_FILENAME.FILE TYPE;VERSION NUMBER

MAES_FC1.ADP;1.1

In this example, 1.1 refers to the file FC1, which is part of the MK 92 MAES program, and of a type used by Softsell's expert system shell, Adept. The particular file is version 1.1.

The identifier for source code is generated when the developer first creates a file. Only the version number is altered from that point forward. An additional precaution, is to include the file identifier (i.e., MAES_FC2.ADP;1.5) in the first line of the program source code, if possible. This protects against the inadvertent renaming of a file. A disadvantage, comes in the form of increased overhead, and therefore, resources. The file identifier must be inserted into the code either manually or through the use of an automated development tool. The cost of inserting the file identifier into the code as a header should be weighed against the need for the additional identification it provides. (Buckley, 1993)

Object files result from the compiling of source code. Object file identifiers remain similar to source code identifiers with the exception of the file type and version numbers. The file type will change from the source code file type to the extension of the

object type (i.e., ".obj" in the case of most compiled languages such as ADA and FORTRAN). An appropriate extension should be identified and standardized across all object files of the same type. The version number for an object file, should be an increment of one higher than the highest version number of the source code from which it was derived. For example, a program whose source code is entitled, TEST_NUM1.C; 1.0 would produce an object file identified as TEST_NUM1.OBJ; 1.1 when compiled. (Buckley, 1993)

Executable files, like object and source files, use the same identifier convention. However, since executable files often contain many different object files, the file names do not normally translate directly to the executable file. (Buckley, 1993) It is important a developer assign a logical file name when an executable file is generated. A discussion of responsibilities for naming files is undertaken later in this subsection.

d. Configuration Identification of Patch Files

The use of code patches is in complete contradiction to the goal behind configuration management. The primary reason is the resulting loss of traceability. However, as Buckley (1993) points out, there are instances in which there is no alternative. Patches should use the same format as presented above. However, they should be clearly identified as a patch, possibly by using the word "patch" as the file name. (Buckley, 1993)

e. Categorization of Software

Identification and control requirements vary between different types of software. For this reason it is important to make a distinction between the various categories of software. By categorizing software, it is possible to establish degrees of identification and control, thereby removing the burden of unnecessary configuration management requirements. This translates directly into reduced overhead, and consequently, controlling the cost of CM implementation.

Not all software that is categorized will require CM. The purpose of categorization is to ensure software which should be under configuration management is

not overlooked. The categorization and relevant CM policies are outlined in the project's configuration management plan. A discussion of the components of a CM plan appears later in this chapter. (Buckley, 1993)

Buckley employs the following categories for software categorization: (Buckley, 1993)

- Category I : Product software. Software under Category I is either an end product, or part of an end product.
- Category II : Software to be embedded in firmware, such as erasable programmable read only memory (EPROM).
- Category III: Vendor-provided software. This software includes all software purchased from a vendor in its "final" form. Operating systems, database management systems, configuration management tools, word processing software and expert systems shells are examples of category III software. The discriminating factor, here, is the vendor's maintenance of this software. If the software is not going to be maintained, either through direct interaction with the developing organization, or through future releases, and the using organization has to assume the maintenance function, then it is not category III software. The category assigned will depend upon the type of software it is.
- Category IV: Test software. Category IV software supports the formal acceptance testing of software in categories I, II, and III. Test drivers, test data, as well as test collection and analysis software are examples of this category.
- Category V: Product-support software. Product support software supports the formal development of category I, II, IV, and VI software, but is not part of the product itself. For example, command files used to compile, link, and load executable files would be included under this category.
- Category VI: Manufacturing support software. This type of software refers specifically to software which supports hardware production, such as control software for a robotic manufacturing device. Category VI software is outside the scope of this thesis.
- Category VII: Other software. This is a miscellaneous category which includes software developed to support informal testing, as well as software that does not fit into any other category.

These categories are examples of a categorization scheme and do not necessarily apply to every project. This is the case with the MK 92 MAES project. For instance, the MK92 MAES project does not make use of manufacturing support software

(Category VI) or software embedded in firmware (Category II). To simplify and tailor the concept of software categorization, the design team adopted a software cataloging system that included software types relevant to the MK92 MAES development environment. The following section discusses the process used

f. Software Categorization in the MK 92 MAES Project

As previously stated, category conventions should be tailored to the specific project. Customizing the categorization process eliminates unnecessary categories. As policies are determined for each category of software, the tailoring of the software categories encourages the establishment of policies that will be customized to meet an organization's needs. The following is a summary of the software categories identified for the MK 92 MAES program.

- Category 1: Product software. This encompasses all files incorporated in a release version of the MK 92 MAES.
- Category 2: Vendor-provided product development software. (e.g., expert systems shells, DBMS software tools, CM tools, etc.)
- Category 3: Vendor provided software not specifically related to product development (e.g., operating systems and word processors.)
- Category 4: Test Software. This consists of any formal test routines written by the development team.
- Category 5: Other software. As the name implies, this is a catch-all category for software not falling into any of the above categories.

g. Configuration Item Naming Responsibility

The assignment of file names should be delegated to the lowest management level practicable; ideally, to individual developers. However, given there is often a need for more stringent procedures, the following procedure is suggested.

The responsibility for naming software configuration items is generally broken down by software category. The categories which are used in the following example are those adopted for use in the MK 92 MAES development effort. The need for

standardization cannot be over emphasized. A lack of discipline in naming the various configuration items would lead to a breakdown of the CM process (Buckley, 1993).

Naming conventions for category 1 (Cat1) software are typically established by a project office configuration management instruction or policy. This policy should be formalized and in writing. The reasoning behind this is the need for consistency throughout the project development team. The instruction would generally only specify the identification scheme and the first part of the name. The originators would complete the name in accordance with the established identification scheme. (Buckley, 1993)

For vendor provided product development software, the vendor's naming convention (e.g., ADEPT 2.2) should be adopted. Typically this will be the vendor's product name, followed by the version number.

Category 3 (CAT 3) software, likewise, uses the vendor's naming convention.

Category 4 (CAT 4) software names are generated by the originator of the software following the established procedures of the project team. The degree to which this software will be brought under CM will depend upon its intended use.

Category 5 (CAT 5) software, as was the case with CAT 4, shall have names generated by the originator. Unless this software is likely to become involved in some way with software production it will not generally be brought under configuration control.

The approach used in naming conventions is largely determined by the degree to which control is desired. The greater the control required, the more formal the naming conventions will be. To minimize overhead, a tradeoff between standardization and an increased burden on the programmers should be considered when drafting CM policy.

h. Configuration Identification of Storage Media

The term storage media includes such technology as CD ROM, floppy disks, hard disk drives, magnetic tape, and optical storage media. Storage media is

generally classified as fixed or removable. Though one might have a requirement to keep CM control over fixed electronic media such as disk drives, this thesis is concerned with removable storage devices. Fixed media is usually controlled as hardware and therefore is not examined (Buckley, 1993).

All moveable electronic media should have a label attached as practicable.

An electronic media label should include the following information: (Buckley, 1993)

- name of the contents (system ID, computer program abbreviation, and the computer program ID number)
- date prepared
- name, telephone number, mailstop, organization of preparer of the item
- number of the version description document (VDD) that specifies the media's contents

By affixing a label containing this information, both the user and the developer have a starting point when attempting to identify a fault.

i. Version Description Document

When software is stored on electronic media in a format for shipment or archival, it should be accompanied by a document containing a detailed identification of the software. This document, in configuration management terminology, is known as a version description document (VDD). (Buckley, 1993)

The purpose of a VDD is to document the contents of all removable media with the exception of vendor provided software, and backup tapes (Buckley, 1993). The master and all copies of all media falling under CM should have an accompanying VDD. VDDs serve as description sheets of the storage media. The following is a list of the typical contents found in a VDD (Buckley, 1993).

- The scope of the VDD: What version does it cover of what program?
- The location of master copies
- Method of preparation: What compiler was used? What was the operating system? Was a particular version of a build program such as Make or Installit used?

- List of included files
- Changes installed: Include a description of the changes made since release of the previous version
- Interface compatibility: State whether or not other programs are affected by this version.
- Reference documents: List the applicable specification document to which this version can be traced.
- Installation instructions: self explanatory
- Possible or known errors: self explanatory
- Signature of person creating this particular copy

j. Configuration Identification of Documentation

Documentation, like software, should be placed under configuration management. Before this can happen, though, it needs to be identified. CI of documentation involves the collection, cataloguing, and labeling of design documentation, specifications, test plans, and associated paperwork. (Buckley, 1993)

Documents are identified using document identification numbers. In addition to the title, this number represents a unique identifier for not only the type of document, but also the specific version. Buckley recommends the use of a fixed identifier, followed by a number which is initialized at one and incremented upon each revision. The following is an example of such a numbering scheme.

MAES-REQ-001

MAES represents the program name; REQ refers to the type of document, and 001 is the version of the document.

This scheme will need to be tailored to the type of documentation, and the degree of granularity one wishes to establish in the documentation identification process. Chapter VI contains an example of a suggested method for identifying knowledge documents associated with the MK 92 MAES project.

2. Establishing Baselines

Baselines are starting points. From these, all development work flows. In configuration management, baselines are established to act as reference points for changes. All changes are applied to a particular baseline, establishing traceability in the process. The following are descriptions of the various baselines. (Buckley, 1993) (Berlack, 1992)

a. Functional Baseline

Functional baselines contain descriptions of the functional characteristics of the system under construction. The included documentation establishes the technical characteristics of what the system is supposed to do. The functional baseline is the first to be established, usually resulting from contractual agreements between the customer and developer. From it, all other baselines are derived. (Buckley, 1993)

b. Allocated Baseline

An allocated baseline can be thought of as a functional baseline broken down into subsets. Allocated baselines describe a configuration item's functional and interface characteristics apportioned from a higher level configuration item. Included in an allocated baseline will be the interface requirements, design constraints, and required demonstration to verify achievement of functional characteristics. (Buckley, 1993)

c. Developmental Configuration

The developmental configuration (DC), unlike the other baselines, is unique to software development. It consists of the documentation and code which describes the evolving configuration of the software under development. The DC is a snapshot of the development at any particular point in time. (Buckley, 1993)

Buckley (1993) describes the developmental configuration as "an internal baseline established to facilitate control of internal developmental activities." It is usually established incrementally as each section of the software is reviewed and agreed upon. The developmental design configuration is broken down into three principle categories. (Buckley, 1993)

- top-level design documents
- detailed design documents
- code

d. Product Baseline

The product baseline is the last baseline established in the development process. It contains all documentation, code, and media required to ensure the software can be reproduced and maintained. The code, media, software tools, and documentation are formally encapsulated in the product configuration and presented to the customer upon acceptance of the product. (Buckley, 1993)

3. Configuration Change Control

This subsection describes the task of configuration change control (CCC). Configuration change control (CCC) is the process by which changes to software are proposed, evaluated, and implemented in a formalized manner. In all to many projects, changes are made indiscriminately, without consideration of the economic, schedule, or other impacts the changes will have upon the system. By establishing a CM process which includes change control, discipline is established in the software development environment. (Buckley, 1993)

a. Configuration Change Control Activities

The Software Technology Support Center (STSC) has identified the following functions as configuration change control activities (STSC, 1994):

- Define the change process
- Establish change control policies and procedures
- Maintain baselines
- Process changes
- Develop change documentation
- Control product release

Though this is an excellent summary of the overall CCC task, a more rigorous look at the change process is necessary if the reader is to understand the rationale behind the CM implementation recommendations made later in this thesis. For this reason, the following change process, proposed by Buckley (1993) is presented.

Buckley (1993) identifies three primary tasks within the change process:

- Identify the problem
- Determine the appropriate course of action
- Implement the change

b. A Comment on the Configuration Change Control Process

The recurring theme throughout the CM process, and particularly true in the case of configuration control, is scale. No two organizations are alike, and no two projects within an organization are alike. Some large software development organizations will have dedicated CM sections staffed with full time personnel whose efforts are directed at the implementation of configuration management. However, CM responsibilities for smaller organizations are often a collateral duties assigned to programmers and project management. Therefore, one should keep in mind, it may be necessary to tailor the CM process presented here, as well as the various documents and reviews, to the needs of the reader's particular project.

c. Identify the Problem

Identification of the problem is the first task which must be undertaken for the change process to be put in motion. This involves three functions. The first step is the documentation of the problem, ideally at the time of occurrence, by the discoverer of the problem. This is typically done on a problem report (PR) or software trouble report (STR). How extensive the problem report should be is dependent upon project size and desired level of detail. (Buckley, 1993)

Just as there are different problems, there will be different environments in which a problem will occur. Taking this into account, it is important the PR be tailored to the needs of the activity identifying the problem. For instance, the information one might

desire or expect from a verification and validation activity would be different from that of a field activity actually using the software. This could require the development of several variants of problem reports or a problem report which encompasses information pertinent to several environments.

The focus of the problem report should be to answer the question, "What happened?" For example what applications were being run? What were the symptoms? What was the configuration of the hardware? What other software was running at the time? At specifically what step did the problem occur?

The next task in identifying the problem is to review the problem report. The number of reviews a report is subject to depends upon the size of the project. One might wish to have a review for clarity of the problem, often referred to as an initial review, before a more thorough review for substance. However, in a small project, there may be insufficient resources for this to be practical. (Buckley, 1993)

Occasionally, what appears to a user to be a problem is not a problem at all. It may be that the user misunderstands the application. If no problem is found, the person who initiated the problem report should receive feedback with an explanation appropriate to their level of technical expertise. The explanation given a programmer may be far more complex than that provided to a user.

If the initial review determines the problem report has identified an actual problem, a priority is given to the problem. Resources are then allocated to determine its cause. Buckley (1993) recommends using due dates instead of priority ratings such as high, medium, and low to establish the problem's level of urgency. He argues the use of a nominal rating system invariably leads to inflation of a problem's priority, thereby corrupting the process. (Buckley, 1993)

In addition to the use of dates, a process should be put in place to identify, evaluate, and correct problems of a critical nature. For example, problem reports identifying problems which have a potential to cause bodily harm or extensive damage to property need to receive particularly close attention. The goal however, is to introduce a

process which accurately prioritizes all change requests, including emergency ones. The CM process should not be discarded, just because an emergency change is required. Instead, the problem should be flagged as being of high priority, and an appropriate due date should be assigned.

The third and final step in identifying the problem is to determine its cause. At this point, the allocated resources set aside during problem review are applied to the identification of the cause of the failure. This could be done through attempts to recreate the failure, code reviews, diagnostic tools, or other appropriate means.

At this point a word of caution is in order regarding the use of problem reports. It must be kept in mind the information a problem report provides is based upon the perceived symptoms of the person who noticed them. In other words, the report may be inaccurate. (Buckley, 1993) Training personnel to properly and accurately fill out a problem report is a challenge and must be undertaken.

d. Determine Appropriate Course of Action

Buckley (1993) recommends the following courses of action be taken depending on the type of problem identified. If the problem results from software which fails to meet its requirements and has a detrimental effect, then the problem should be corrected so that it is in accordance with the specifications. (Buckley, 1993)

If software departs from the specified requirements but can be used as is with no detrimental affect, then Buckley recommends initiating a request for waiver. A decision can then be made as to the need for a change to future releases (Buckley, 1993)

If the software's specifications are to be changed and the customer's approval is required, then a change proposal should be initiated. (Buckley, 1993)

If the specification is to be changed, either after customer approval or if customer's approval is not necessary, then a specification change notice should be initiated, and change to the specification should be made. (Buckley, 1993)

The decision on how to proceed is formally decided by a configuration change board (CCB). This board, its purpose, and make up are discussed in a later subsection of this chapter.

e. Implement the Change

Implementing the change to software ultimately falls upon the development or maintenance team. This involves not only the physical recoding of the software, but also the follow-on and regression testing, verification, and validation. Before a change can be considered to be complete, it must be tested for correctness and compared to the specifications to ensure the change has been implemented as intended. Although the specifics of testing, verification, and validation are beyond the scope of this thesis, the interested reader is referred to the thesis by Dills and Tutt (1994) for further discussion.

Figure 3-2 is an example of a generic software configuration change process. (Berlack, 1992) (STSC, 1994)

f. Configuration Control Board

The fundamental purpose of a configuration control board (CCB) is to provide a clearinghouse for all changes made to baselined configuration items. The size and makeup of the configuration control board will be dependent upon the complexity of the project, and the number of personnel available. The CCB should include those personnel who are involved in the identification of the problem, if practical. It includes those responsible for taking the corrective action; the personnel in the best position to assess the impact of the problem; and the configuration manager. (Berlack, 1992)

The degree of formality is ultimately dependent, upon the type and scale of the project in question. Some projects, such as those responsible for developing flight critical software for the space shuttle, have extensive configuration control processes, with several configuration control boards and strict, formalized procedures. This need not be true of smaller projects. The goal is to develop a configuration control board which will include the appropriate level of attention for the problem, in such a manner as to provide controlled, rapid, and decisive feedback to the development process.

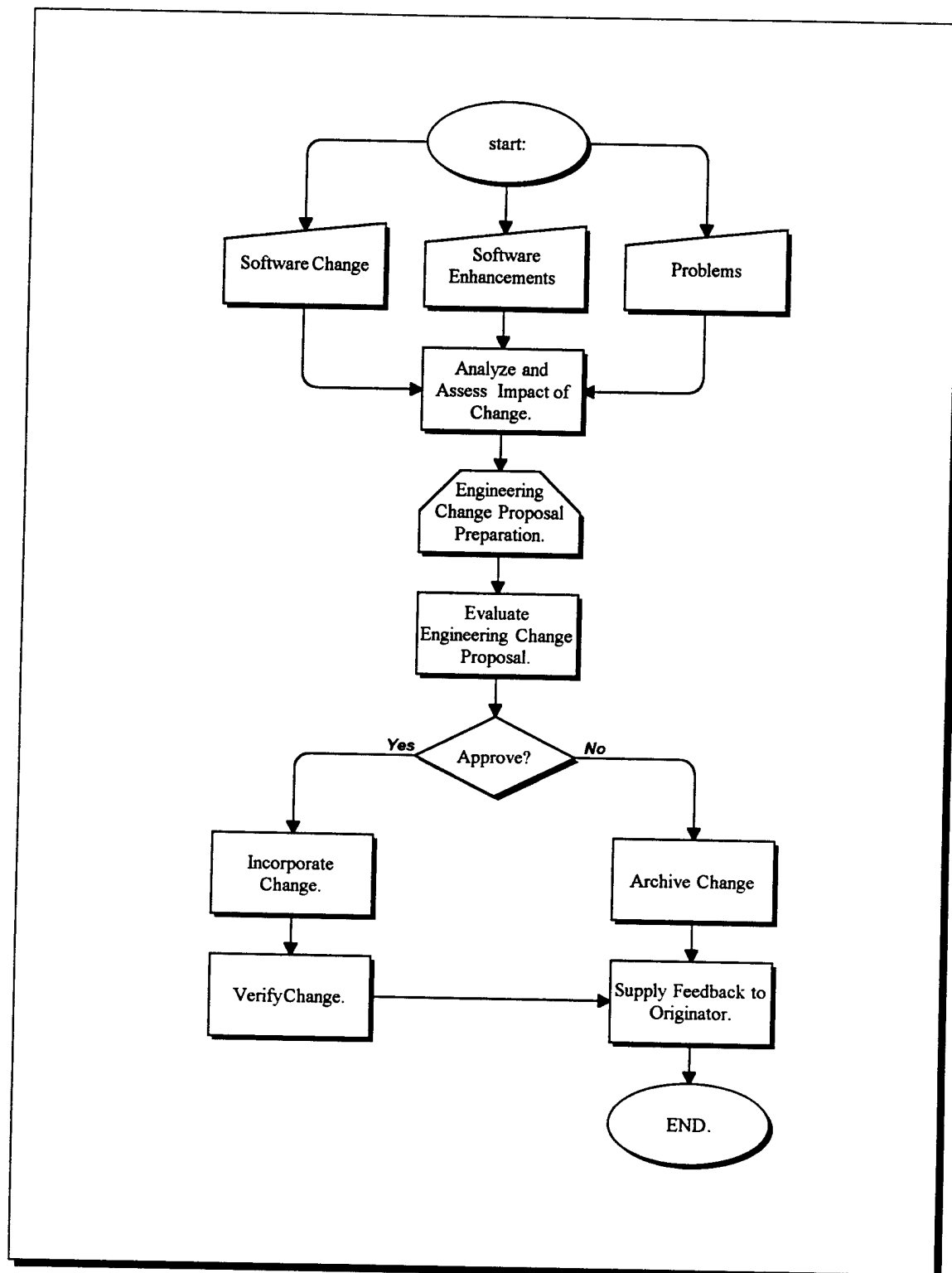


Figure 3-2. The Change Control Process.

The software configuration control board acts as a decision point. A change proposal will come before the board for approval. At this point several things could happen. First, the board could determine there is no change needed and fail to approve the proposal. If this is the case, the proposal should be archived, the status of the change should be closed, and feedback should go back to the initiator of the change. Another scenario is that the board may recognize the need for a change, but due to an analysis of its impact, it may determine not to incorporate it until a later release. In this instance, the status of the change should be left open or active. Feedback should be sent to the initiators of the change informing them of its status.

The final alternative is to incorporate the change in the current development effort. If this is the decision reached, the change proposal should be turned over to development. As always, the initiator of the change should receive feedback.

Within each of these decisions are a couple of common threads. The first is the need to provide feedback. Problems with implementation need to be identified. By providing feedback to the person who initiated the problem report, he/she is encouraged to continue pointing out potential discrepancies. Such communication supports the aim of consistently striving to improve the software and thus producing a quality software product.

The second common thread is the need to archive and track all potential changes to software configuration items and their associated documentation. This strikes at the very heart of CM. As Chapter V illustrates, there are tools in the marketplace that address the challenge of change control and the CM process.

g. Change Control Authority

Just as there is a control over what changes are made, there is also a need for control over who can authorize changes. This authority is referred to as change control authority (CCA). This authority is typically organized and granted by baseline. The following is a list of the baselines and who is usually granted change authority.

Again, as is the case throughout this thesis, CCA must be tailorable to a given project's specific situation. (Buckley, 1993)

- **Functional.** Usually, the customer's approval is required to change the functional baseline.
- **Allocated.** Changes to the allocated baseline are approved at the project management level. Typically approval is granted by the project manager (PM).
- **Developmental Configuration:** Changes to the developmental configuration are also approved by the project management, usually the PM.
- **Product Baseline:** Approval of changes to the product, or deliverable is a result of negotiation between the customer and developing organization. The approach to product baseline approval outlined here is geared toward software projects (such as those in DOD) in which a development organization is under contract to produce software for a customer. The approval authority for decisions to create a product baseline and make subsequent changes to it must be established in the contract. This policy would not apply to a private company such as Microsoft and their decision to release a product. A software vendor making products for general distribution (i.e., Microsoft Windows, or Borland C++) grants approval for changes independent of customer involvement.

h. Configuration Change Control of Specific Software Categories

The configuration change control of software varies for each category of software. The degree of control placed upon software that is in a developer's working directory is likely to be much less than that of product software. Therefore, it is necessary to discuss some of the common practices when establishing CCC over a particular configuration item.

The degree to which CCC is applied to CAT 1, product software, will depend upon the stage of development the concerned software is in. Early in its life cycle, before the software is made available for other programmers to work with, very little change control is required. It is not until a particular software item is tested and/or used in conjunction with other items that configuration control becomes more stringent.

When a software item is first developed, changes, and consequently change control, are the responsibility of the programmer. Once a module, file, or other item is completed to the point it is successfully compiled, presented for testing, and/or included

with other items for other developers to use, configuration change control need to be initiated. A litmus test to determine whether an item should come under CCC is to ask the following questions:

- Is there any possibility of another developer other than the originator gaining access to this work, even though it is "in progress"?
- Will this work be included with other items for testing?
- Will this item be included in a library or directory where the potential exists for others to use it? (Buckley, 1993)

If the answer to all of these questions is "no," then there is no need to place the software under CCC. Instead, it should be considered work-in-progress.

Once software is moved to an area in which others may access it, or introduced for testing, configuration control responsibility must move to a higher level. Why? Consider the possibility of a programmer who, after introducing code for testing, continues to make unrestricted changes. Other programmers download the version introduced and use it to develop their code. If control was still in the hands of the originator of the code, other developers might discover, as changes are made, code which once functioned properly, no longer does. To avoid this, it is important for the developer to pass configuration change control to the next level of management. (Buckley, 1993)

As product software passes succeeding levels of testing, verification, and validation, its configuration change control becomes more tightly controlled. Ultimately, as final acceptance trials are completed, change control authority will pass the product into the change control process. (Buckley, 1993)

Vendor software, CAT 2 and CAT 3, presents more of a challenge. Often, a development organization is not in a contractual position to have any control over the vendor's configuration management process. Ideally, the vendor has recognized the benefits of CM and has instituted their own program. Unfortunately, as Capers Jones (1994) points out, many have not.

So what role does configuration change control have in the CM of vendor software? The degree of CM control ultimately depends upon the criticality of the

software under development. If the product software is mission critical whose failure would result in loss of life or substantial property damage, then several options exist:

- Establish a requirement for vendors to implement CM programs if their software is to be used in conjunction with the product software. This is what NASA has done with space shuttle software.
- Contact the Vendor and explore the possibility of taking those modules of the vendor software which are critical to the operation of product software and placing them under the development team's CM program. Due to licensing, trade secrets, and copyright restrictions, this may not be feasible. However, language could be included in the agreement that would permit such a process while ensuring the sanctity of the vendor's intellectual property.
- Control changes to vendor software at the version or release level. In this method, the decision to change to a later version of a vendor's product already under configuration change control would be made by the CCB. After an assessment of the new version's impact, the decision would be made as to whether or not to include it in the project's configuration library. This may be the most economical approach for most software development projects.

Why are we concerned with placing vendor software under configuration change control? In most cases, as changes to vendor software occur, the backward compatibility with earlier versions is unaffected. However, this is not always true. Therefore, to prevent compatibility problems between code developed under previous versions and those developed using the current version, it is necessary to place the development software under CCC.

Another concern is the situation in which a project team is using development software from multiple vendors. As changes are made to one vendor's product, updates to another vendor's software may not be made for some time, if at all. The potential exists for two software tools which were at one time compatible to become incompatible. Configuration change control of vendor software enables the development team to identify potential problems before damage can be done to the project. (Buckley, 1993)

The degree to which configuration change control is exercised over CAT 4, test software, will depend upon its intended use. If the software is a one-time-only test

program which will be discarded upon completion of its use, then it should be controlled as a software item that will receive no changes. The advantage to this control method, is that the documentation, and formal process for implementing and tracking changes need not be maintained. (Buckley, 1993)

If however, the test software is to be modified or retained for future tests as a reusable software tool, then it needs to be tracked with the same degree of integrity as the software it is testing. (Buckley, 1993)

Cat 5 software will typically not be brought under CM, and therefore is not normally subject to change control. (Buckley, 1993)

i. Configuration Control Documents

In addition to the procedures outlining the CCC process, there are several documents associated with change control. These include the software trouble report (STR), known generically as a problem report; the change request (CR); the engineering change proposal (ECP); and the request for waiver. Each has a key function in the CCC task.

The STR documents a perceived problem for its subsequent review and corrective action. This document frequently initiates the CCC process at such times as when a user's application crashes.

Change Requests are sometimes used in lieu of STRs or problem reports. In some organizations, change requests are specifically geared toward enhancements or changes which are not related to what the customer perceives as incorrect operation of the product. (Buckley, 1993)

Engineering Change Proposals are a product of the identification and review process. Once a problem has been identified, it is sent to the change control board (CCB) for approval. The ECP is a recommendation to the board which includes a summary of the impact a change will have on the product, if known, and a recommended course of action.

Requests for waiver are sometimes submitted in instances where a software product does not function in accordance with specifications, but the discrepancy is not one which would be detrimental if left unchanged. Waivers should only be granted on a temporary basis, and an analysis should be undertaken to ascertain when a change correcting the discrepancy can be corrected.

The degree to which a particular project adopts these documents will largely depend upon the extent to which the change process is formalized and the degree of automation. For those documents adopted by a development team, a process should be initiated which tracks the status of changes and problems reported by these documents. Problem tracking software and document management software are useful in accomplishing this. Where automated methods of tracking the progress of problems and changes, a manual method, such as loose leaf binder or log book may be necessary.

j. Configuration Control Concerns

Buckley (1993) points out the danger of the configuration control process evolving into a bureaucratic process that exists for its own self survival. This is a very real threat that encompasses as many project management issues as organizational behavior topics. There is a need to control the change process, but one does not wish to stifle creativity or flexibility.

The goal is to establish CM as a part of a greater emphasis on quality. By focusing on various aspects of the organization (such as employee training) in addition to those concerned with procedures, the threat of creating a bureaucracy of the change process is somewhat diminished. CM will become one aspect of a larger approach to achieving product quality.

There is also a perceived danger of configuration control slowing down the process. When a layer of overhead is added, it could potentially bog down the process. However, before one dismisses configuration change control in the name of streamlining, consider the consequences of not having configuration control. Developers could be working with a great deal of uncertainty. Changes could be made with no consideration of

the economic impact of their implementation or their effect on the development schedule. The result would likely be a rise in development costs and a project which falls behind schedule.

What would happen if an ad hoc decision was made not to make a proposed change? The failure to apply the necessary analysis to evaluate problem's impact could result in a decision which could lead to fault ridden software with the potential to damage equipment, information, or at the very least, reputations. Decisions not to make changes, just as decisions to proceed with proposed changes, must be based upon a thorough evaluation of the alternatives. They cannot be made indiscriminately by overworked programmers or project managers concerned about schedule delays. When one considers the alternative to configuration control from a systems approach, the idea of a configuration control process no longer seems like such a bad idea.

k. The Need for Automation in Configuration Change Control

If we are to develop a successful configuration control process, it needs to be as simple and seamless as possible. One cannot reasonably expect software developers to follow the rules of a bulky, complex configuration control system. Therefore it is desirable to automate as much of the configuration control reporting system as technologically feasible. Technology exists to ease the impact CCC has upon the every day programming environment. Chapter V provides a more in depth look at how CM tools can assist the configuration manager in establishing and maintaining a CM process.

4. Configuration Status Accounting

a. Overview of Configuration Status Accounting

Configuration status accounting (CSA) is the reporting and recording of the information needed to manage the functional and physical characteristics of configuration items . These reports include a listing of approved documents that describes the physical and functional characteristics of the product software; the status of proposed

changes, deviations, and waivers to those characteristics; and the implementation status of approved changes. (Buckley, 1993)

Configuration status accounting refers to the record keeping activities intrinsic to the previously discussed CM tasks. It also has come to refer to the method of storing the information associated with the software's configuration. This information must be stored in a format that ensures traceability from specification to code. (Berlack, 1992)

CSA serves as an important project decision making tool. Using queries and reports generated through the project's status accounting activity, a manager can determine the progress (or lack of) a development team is making relative to a set time schedule, functionality requirements, or other yardstick. (Berlack, 1992)

Another role for status accounting is in software maintenance. Maintenance begins with the first change and continues for the lifetime of the software. As changes, enhancements, and corrections are made, a history is developed. Frequently, developers have a need to refer to this history to determine the design rationale for code they encounter. Previous changes and problems provide clues for troubleshooters. Unfortunately, all too often, the required information is either nonexistent or poorly captured. Configuration status accounting's aim is to minimize this problem. (Berlack, 1992)

Finally, the history files created by the status accounting task can be used to improve the software development process. By analyzing the changes made, time to complete the changes, as well as other information provided through the CSA process, insight is gained into the maturity of the project team's development methodology. (Berlack, 1992)

b. The Configuration Status Accounting Process

There are two major components of the CSA process: identification of the reporting requirements and the actual recording of the information itself.

Identification of reporting requirements can be accomplished through the examination of two sources. The first is the agreement or contract between the customer and developer. The second is the developing organization's regulations and policies.

Agreements between the developer and customer often include minimum reporting requirements. These reports keep the customer informed of the status of changes, enhancements, and corrections, as well as resources being applied to various aspects of the project. Where applicable, they should be included in the CSA structure.

In addition to those reports required by the customer, there may be reporting requirements mandated by the developing organization's internal policies. Management controls and process improvement efforts require information. Although not the only source of information, reports derived from CSA provide a useful management tool.

The types of reports and queries a project's personnel require will vary depending upon its complexity and maturity relative to the development life cycle. In some projects, developers may employ configuration management metrics such as the rate of changes. In others there may be little regard for such information. It is important here to recognize that the CSA process adopted must take the organization's information requirements into account. As with change control, tools exist which can help provide such support.

5. Configuration Audits

Webster (1977) defines an audit as a "methodical examination and review." As the definition implies, the configuration audit determines whether or not what was required was actually built. Configuration audits apply not only to the code, but also to the accompanying documentation.

a. Types of Configuration Management Audits

In terms of configuration management there are three types of audits; the functional configuration audits (FCA), physical configuration audits (PCA), and in-process

audits (IPA). Each reviews a different aspect of configuration management. (Buckley, 1993)

b. Functional Configuration Audits

During the functional configuration audit (FCA), the performance of a configuration item is compared to its required performance as defined in the functional and allocated baselines (Buckley, 1993). To do this, test output is compared for compliance to the software's specifications. The FCA is for the benefit of both the customer and developer. Therefore it should be conducted in the presence of representatives from both the customer organization and the development team (Berlack, 1992). This technical examination can be a difficult task and may require the auditing team to combine their expertise to come to a consensus (Buckley, 1993).

The FCA is typically held at end of development cycle and after the configuration items subject to audit have been tested. Buckley (1993) cautions against the urge to evaluate the testing program itself. Auditors must bear in mind that the goal of FCA is to test whether or not the configuration item's performance is in compliance with stated requirements (Berlack, 1992).

c. Physical Configuration Audit

The physical configuration audit (PCA) is typically performed after the functional configuration audit has been completed on all configuration items (Berlack, 1992). In a PCA auditors attempt to determine whether or not the design documents, product specifications, and associated documentation are representative of the software that was developed. This is the last step before establishing the product baseline. In instances where there is to be a transition of responsibility, the ownership of the software shifts out of the hands of the developers to the customer. (Berlack, 1992)

d. In-process Audits

The purpose of an in-process audit (IPA) is to evaluate the quality of the configuration management process itself.(Buckley, 1993) Such things as traceability

testing, evaluation of change documentation, and adherence to CM policy are under the scope of an IPA. These audits can be conducted by outside organizations such as the QA organization, or preferably through self-auditing as the software engineering process matures.

e. Conducting Configuration Audits

The first step in conducting a configuration audit is to schedule a time and draft an agenda. The agenda is drafted by the auditors. However, the auditors should request input from the project manager as to the schedule of various aspects of the audit. By contacting the project manager in advance, the auditors are ensured adequate resources will be available (Buckley, 1993). This also enables the software developers to schedule not only the audit, but other activities in such a way as to minimize their impact upon the schedule.

The next task occurs on the day of the audit. A brief should be conducted with the organization being audited. The key personnel involved in the audit should at this point introduce themselves and their role. This establishes points of contact to keep the process flowing smoothly. (Buckley, 1993)

The third task is the audit itself. The audit should follow the established agenda closely, both in terms of schedule and scope. Disagreements should be mediated where possible to minimize the establishment of an environment in which people attempt to hide the truth. (Buckley, 1993)

Upon completion of all audit functions, there should be a brief review of the auditor's findings. Any major discrepancies, as well as any particularly good aspects, should be pointed out to the applicable personnel. A summary of the problem areas should be provided to the project manager if possible, so that corrective action can be taken as soon as practical. A more detailed compilation of the auditor's findings should be delivered later in the form of an audit report. (Buckley, 1993)

Audits, regardless of type, should focus on improvement. All too often, audits are looked upon by developers as another way to be penalized for errors. If true

process improvement is to be gained from the CM audit task, then process improvement needs to be the focus of the audits. Audits conducted as witch hunts will only result in falsification of records and an "us versus them" attitude in the software engineering organization; all of which is contrary to the process improvement goals of the software engineering disciplines.

C. THE CONFIGURATION MANAGEMENT LIBRARY

The configuration management library serves as the central repository for all configuration management items. Various versions of a software project under development are stored in the library as well as all changes and associated documentation. Its existence is fundamental to the establishment of a CM program.

The simplest CM library is a simple database which stores all relevant data associated with a project. However, Berlack (1992) notes the trend toward CM libraries which are made up of smaller libraries. These include working libraries, project support library (PSL), master library, software repository, and backup libraries. The following is a brief description of the libraries used in today's CM implementations.

a. Working Libraries

Working libraries (WL) are directories set aside for individual programmers that are protected from access by other programmers (Berlack, 1992). The desire to place control on the development process as early as practicable is the reason one establishes a working library for a programmer's work-in-progress. Working libraries, although not subject to rigorous configuration management, do promote CM by shielding the work of one programmer from another. As barriers, WLs prevent the overwriting, deletion, or duplication of untested software.

b. Project Support Libraries

When a programmer wants to work on a particular configuration item or file, it is extracted from what is known as a project support library (PSL). When

completed, the item is returned to the PSL for testing or storage until needed at a later time.

In addition to storing files which a programmer is not working on, the PSL acts as a staging area for the integration of modules. By collecting files in the PSL for integration and testing, the integrity of the master library is maintained. (Berlack, 1992)

c. Master Library

The master library is the storage area of all configuration items which have been tested and placed under more stringent CCC than was established under the PSL. Because the master library is the baseline library for the project, it is typically under extremely tight access control. Such control is essential if software integrity is to be maintained. (Berlack, 1992)

d. Software Repository

The software repository can be considered to be an archive. Through the establishment of a repository it is possible to reconstruct previous versions of any configuration item under control. Software repositories are useful for troubleshooting and production of a change history. (Berlack, 1992)

e. Backup Library

The backup library should contain copies of all libraries which are critical to reconstructing a particular version (Berlack, 1992). The establishment and update of a backup library is essential if an organization wants to ensure some capability of disaster recovery.

1. The CM Library in Operation

Since the CM library is central to a successful configuration management program, the reader must be familiar with its operation. For the purposes of this thesis, all references to CM library principles shall be made with a CM library established in electronic media.

a. Check-in Check-out Concept

The check-in/check-out concept refers to the process by which programmers fill their working libraries. In operation, programmers would check the files on which they wished to work out of the program support library. When finished, the developer would return the updated version to the PSL. This ensures the developer is always working with the most recent version of the code under construction. (Buckley, 1992) (Berlack, 1992) Tomayko, 1990)

b. Migration

Migration is a term which describes the method of moving various items through the various life cycles of a project in terms of the CM library. As items are developed, they are checked out by test personnel who then return the item to the library for more work or formal acceptance. As the item is shifted from the PSL to the master, or other library established as part of the software development life cycle, it is migrated (Softool, 1994a). Migration should be under strict access control, as the integrity of the master library or other libraries could be jeopardized.

c. Change Regression

Change regression refers to a situation in which the changes made by one programmer are overwritten by the work of another programmer. For example, two programmers, A and B, each check out the same version of a configuration item, X, to make some changes. Programmer A checks the item out before programmer B completes the change and returns it to the library. Programmer B decides there is not enough time to finish the needed changes. The unchanged version of X is uploaded by programmer B, overwriting the changed version checked-in by programmer A. The end result is a problem in which the changes implemented by A are signed off as complete; however, in reality the actions of programmer B have erased any work accomplished by programmer A. The problem of two people working on a file at the same time is often referred to as

the simultaneous update problem. (Softool, 1994a). (Tomayko, 1990) (Berlack, 1992) (Buckley, 1993)

d. Promotion

Promotion is a term that describes the action of incorporating software items that have been tested into a final version for release (Softool, 1994a). Promotion should have the highest access controls of any of the previous functions, as it has the greatest impact on the software being delivered to the customer. Promoted items become the new product baseline from which future changes are made.

e. The Single User Approach to Using a CM Library

Figure 3-3 (Buckley, 1993) is a diagram of the single user process. Because only one user is drawing from and adding changes to the PSL many access controls which would be incorporated in a multiple user environment are unnecessary. Primarily, only migration and promotion access controls are mandatory, though check-in check-out is still desired to maintain a change history. (Buckley, 1993)

f. The Parallel Development Approach to Using a CM Library

The parallel development environment is where one encounters the simultaneous update problem. Because of the dangers to software integrity, more restrictions need to be put in place for a parallel development CM library. In addition to migration and promotion access restrictions, check-in check-out becomes mandatory in order to know who made what changes.

To ensure each developer is using the latest version of an item, a lock out feature is incorporated in the CM library. A lock out feature, as the name implies, prevents a configuration item from being worked upon simultaneously by two or more programmers. This aids in preventing change regression. However, unless there is adequate communication, testing personnel may not know whether or not the version they are extracting from the PSL has incorporated all outstanding changes. Configuration

change control and status accounting, if properly implemented, will be able to resolve this dilemma by identifying the status of all changes.

D. CONFIGURATION MANAGEMENT KEY PERSONNEL

It will become apparent that virtually all people associated with the project are in some way responsible for some aspect of CM. In short, CM can only be successful with the participation of everyone involved with the implementation of a software product.

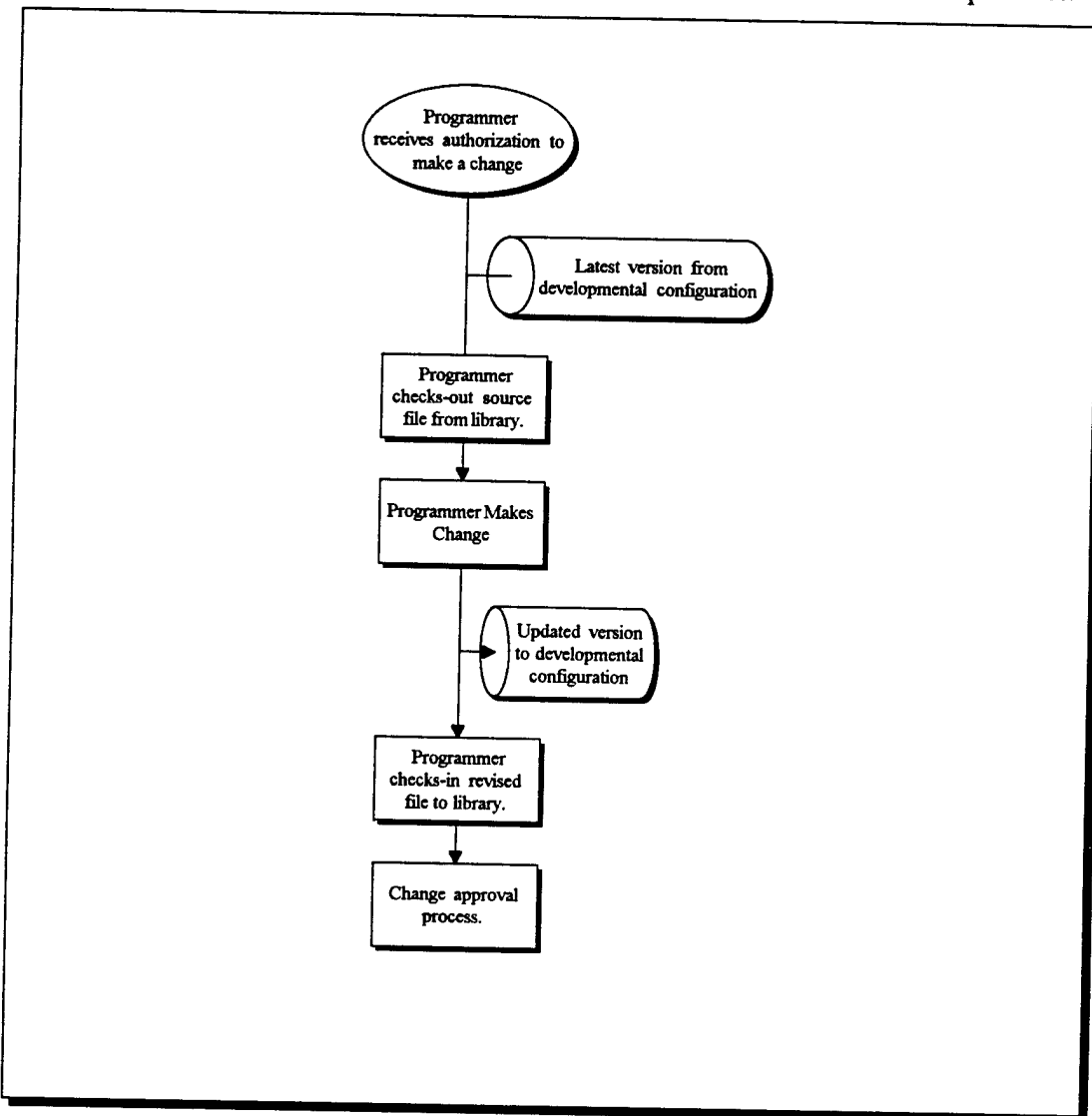


Figure 3-3. Example of a single user process for working with a CM library from Buckley (1993)..

1. The Customer

The customer's role in CM cannot be overstated. Whether one discusses their role in establishing requirements, participation in the auditing process or feedback regarding desired changes and problems, the involvement of the customer is crucial. As stated earlier, CM does not end with development. It is a key ingredient to maintaining software, long term, in a cost effective manner. If the customer is going to take on the responsibility of maintenance, they must be an integral part of the CM process. They may wish to transfer the developer's CM process into their own management process.

2. Configuration Manager

The configuration manager is responsible for the proper operation of all aspects of the CM program. Typically this person answers directly to the project manager on all issues related to CM. As the keystone of the CM process, the CM manager must be intimately familiar with all aspects of configuration management, as well as emerging trends in CM implementation. A more thorough treatment of the responsibilities of a configuration manager can be found in Chapter VI.

3. Configuration Management Library Administrator

The CM library administrator (CLA) is responsible for the maintenance of all facets of the various configuration libraries. A CM program revolves around its CM library. For this reason the CLA's role is as important, as that of the configuration manager to a successful CM process. A more detailed review of the duties and responsibilities of a CLA can be found in Chapter VI.

4. Individual Developers

The bulk of the responsibility for a successful CM program falls upon the shoulders of the developers. On a daily basis, they are the ones implementing CM policy. Developers write many of the STRs and provide feedback as to the adequacy of CM tools and processes. Their adoption of the CM philosophy is critical; otherwise change

documentation will exist, but not be thorough. If developers are not supporters of CM, procedures will be circumvented until the result is a CM process in name only.

5. Project Management

The only way configuration management will become an integral component to the SDLC is if continuous management support is obtained. At the outset, CM requires resources such as software, training, and personnel. Those in control of those resources must be behind the concept of CM to assure success.

As the CM process matures, management's involvement will promote the improvement of the process, as well as ensure developers are adhering to established policy.

6. Configuration Control Board

The role of the configuration control board in controlling changes to software has been previously discussed. In addition to their duties associated with the mechanics of CM, the CCB also has a responsibility to provide quality assurance to the CM process. The CCB is the point where change requests, impact analysis, and other aspects of project management collide. It is their responsibility to ensure changes supposedly already made, aren't reappearing, and that the quality of the documentation and impact analysis is maintained. The CCB is in the position to see problems before they develop. As important as their role in change control is, their role in process management is equally important.

E. THE CONFIGURATION MANAGEMENT PLAN

1. What is a Configuration Management Plan?

A software configuration management plan is a document that sets forth guidance for the identification of configuration items, controlling and implementing changes, identifying reporting procedures, and establishing auditing responsibilities (IEEE, 1983).

A CM plan is a blueprint for establishing a successful configuration management program. It establishes, in Buckley's (1993) words, "a common method of doing

business." In addition, he recommends a separate CM plan for each software project an organization undertakes. When one considers the different requirements, customers, and types of projects, this advice is well taken.

The purpose of this section is to highlight some of the key aspects of a CM plan. Chapter VI presents some recommended procedures for implementing various components of configuration management for the MK92 MAES.

2. Developing CM Plans

Bounds and Dart (1993) identify the configuration plan as one of three key components to what they call a successful configuration management solution. The other key facets are the CM tool selected and the adoption strategy. (Bounds & Dart, 1993). A discussion of CM tools can be found in chapter V. An initial recommendation for an adoption strategy for the MK 92 MAES project is presented in Chapter VI.

The development of a CM plan is no small undertaking. The goal in designing and developing a CM plan is to describe and document the configuration management policies of a product as early as possible in its life cycle (Berlack, 1992). The framework of the plan itself may be written in a short period; however, writing the procedures and policies which will implement the plan is significantly more difficult (Bounds & Dart, 1993). Furthermore, the CM plan, like the software process it supports will evolve. This requires a periodic review of the CM plan to ensure it still fits the development effort's needs.

3. Guidance on Configuration Management Plans

Configuration management plan guidance may be found from several sources:

- CM plan standards
- CM plan templates found in some CM tools
- CM plans from established CM programs

Bounds and Dart (1993) conducted a survey of personnel involved in configuration management, one portion of which was devoted to the applicability of NASA, IEEE, and

DOD configuration management plan standards to real world projects. Each was compared on six traits:

- ease of use
- completeness
- tailorability
- consistency
- correctness
- life cycle connection

Each trait could be assigned a score from zero to three with three identifying a standard which "satisfies requirements for excellent standard." (Bounds & Dart, 1993). The IEEE standard received a score of three in every category save completeness, where it received a two, and life cycle connection, where it received a one. Neither the NASA nor the DOD standards came close. Each had only one three, both in the trait of completeness, and they likewise received ones for life cycle connection. In summary, the IEEE standard was overwhelmingly preferred over the NASA and DOD standards. Therefore, the CM plan outline provided in the IEEE standard shall be the example from which the MK92 MAES CM plan is derived. For further reading, Appendix A lists standards and guidance which are applicable to CM and the development of CM plans.

Configuration management tools are beginning to come with CM plan templates that can be customized to fit the needs of individual organizations (Bounds & Dart, 1993). Before selecting a CM tool for its template, one needs to look at the value of all the tool's features to the project. A good template may lead to a great looking CM plan, however, a poor tool will kill the program the plan was attempting to implement.

As for the use of CM plans, Berlack (1992) warns against the blind copying of an existing organization's plan. Their process may be outdated, or inadequate for the needs of the prospective CM plan author's organization. However, he further adds, if the plan appears to be well suited to your organization, change it to fit, and use it.

4. The Components of a CM Plan

This subsection presents a summary of the primary features of a CM plan. A configuration management plan is typically divided into three sections.

The first section is typically one which provides an overview of the CM plan. The specific purpose of the plan is outlined. Additionally, this section identifies those items, organizations, activities, and life cycle phases which fall under the CM plan guidance.

Section two of a CM plan establishes the management of the CM program. The functional organization is established as well as the responsibilities of personnel and agencies falling under its scope. Implementation issues, applicable policies and regulations, and interfaces between components are also specified.

The third section deals with the CM tasks in greater detail. The items which are to be baselined are established, as well as the change control process. Both internal and external reporting requirements are established for configuration status accounting. Additionally, auditing items are identified and a process stipulated for configuration auditing.

Additional sections could include guidance on the use of particular CM tools, procedures for the CM of vendors and subcontractors, as well as records collection and retention. As an individual CM plan should be established for each project, the contents will vary from plan to plan. The goal is to establish a process which is manageable for a given scale of project, while retaining flexibility for growth as needs dictate.

5. Summary of Configuration Management

Configuration management plays a key role in the software improvement process. It controls and tracks changes, provides version control, and serves as a management tool. Through the tasks of CM, identification, change control, status accounting, and auditing, an organization can implement a product integrity supporting process that is repeatable. This not only will reduce costs, but will also improve productivity and time to market; all goals of not only the private sector, but also DOD.

IV. ISSUES IN CONFIGURATION MANAGEMENT OF EXPERT SYSTEMS

This chapter addresses the issues related to the application of configuration management to the development of expert systems. The chapter begins by exploring general issues surrounding expert systems. This is followed by proposing a novel approach to the application of CM to an expert system's knowledge base. Finally, the chapter concludes by examining the issues affecting the CM of expert system implementation.

A. GENERAL CM ISSUES OF EXPERT SYSTEMS

The move of expert systems from the realm of academia to mainstream applications in business, engineering, and DOD presents along with the potential benefits, reason for concern. McCaffrey (1992) points out,

While hundreds of expert systems have been fielded, almost all have been designed and developed without serious thought being given to their long-term maintainability.

A vast majority of literature regarding expert system development centers on the technical aspects of knowledge acquisition, representation, and implementation. Even fundamental works on the subject of expert systems such as Prerau (1990) and Walters and Nielsen (1988) devote only a couple of pages to the concept of expert system maintenance. While these works recognize the need to address maintenance issues, they offer little guidance on the effective application of maintenance principles. Configuration management, a discipline which establishes control over the change process, has largely been ignored.

"In order to effectively maintain an expert system," Bielawski and Lewand (1988) state, "the developer must establish detailed criteria for modification of the system." Although this is generally true of any software, it is particularly so for expert systems. In addition to the need for configuration item identification, version control, and change control of the software, consideration must also be given to maintaining the knowledge

base. The frequency with which the knowledge base changes, and the difficulty of assessing the impact of such changes further complicates the maintenance of expert systems.

The following is a list of challenges that are particular to applying CM to an expert system development effort.

1. Lack of Detailed Specifications

A lack of well defined specifications is one problem complicating the application of configuration management to expert systems. Traditional programming development cycles include the establishment of detailed specifications to which all further development is traced. This is not the case with expert systems (Sacerdoti, 1991) (Prerau, 1990). As Prerau (1990) emphasizes, the domain expert's knowledge serves as a surrogate for the specification of the expert system under development. Therefore, the specification and implementation evolve simultaneously. The nature of an expert's system development process makes traceability and the entire CM process more difficult.

2. Difficulty in Identifying Baselines

Chapter III identified the configuration management baselines used in traditional software CM. Although similar to traditional software, the lack of functional specifications prohibits the establishment of a detailed functional baseline at the outset of an expert system development effort. When applying CM to traditional software development efforts, the allocated baselines, developmental configuration, and product baseline all evolve from the functional baseline. Lacking functional specifications, a different approach to baselining is necessary.

a. The Functional Baseline

Although similar to traditional software implementations, the functional baseline of an expert system will not be as well developed as that of a traditional program. Initially, only documents describing the desired expert system's capability in broad terms may comprise the functional baseline of the expert system. As knowledge is added and the

expert system is developed, the functional baseline will evolve. It is not until the product baseline for the expert system is first established that its functional baseline is completed. (Prerau, 1990)

To establish a baseline for configuration management purposes at the project's outset, documentation which describes the general functionality of the expert system should be considered to serve as a functional baseline. This could include, clauses in the contract, a statement of work, concept documents, or other documents describing the final product.

b. The Allocated Baseline

The existence of an allocated baseline will depend upon the existence of documents which detail any interfaces or other subset of the functional baseline. If interface description documents exist, then allocated baselines may exist for the knowledge and interface requirements. If no such interface requirements are stipulated, the creation of an allocated baseline would be unnecessary.

c. The Developmental Configuration

This is the documentation and knowledge which represents the expert system as work in progress. Since the knowledge is evolving throughout the expert system's development, a snapshot of the developmental configuration's knowledge will define the current functionality of the expert system.

d. The Product Baseline

As with traditional software, the product configuration includes all code and documentation which allows for the production of the expert system. However, in addition to the code and related documentation, the product configuration includes the approved representation of the knowledge base. This knowledge defines the functionality of the expert system, and therefore is used as the functional specifications for the functional baseline.

3. CM Challenge Posed by Prototyping

The lack of detailed specifications before implementation precludes the use of the traditional top down approaches to software development (Prerau, 1990) (Sacerdoti, 1991). Instead, expert system developers tend to use an incremental, or prototyping approach to their implementation. Unfortunately, maintainability is not of particular concern to expert system developers as they attempt to implement an example of the desired functionality in the form of an initial prototype (Prerau, 1990).

While prototypes of traditional applications are often discarded, the prototyping of expert systems is often different. Even though it is only a prototype, the software, through the natural progression of the expert system development cycle, must be mapped from knowledge which has already been elicited from the expert. Though this knowledge might be discarded as well, it will more likely evolve into the product's knowledge base.

Likewise, the product itself, will evolve rather than be re-implemented, particularly if an expert system shell was used to create the prototype (Prerau, 1990). Unfortunately, the need for configuration management is often not realized until a problem calls for its use. Implementing a CM process in a pre-existing project is far more difficult than planning the CM process from the outset (Buckley, 1993) (Berlack, 1990).

Prototyping implies the rapid creation of versions for evaluation. It is possible many versions of an expert system could be in circulation at any one time. This complicates the CM process due to a need for version control utilities earlier in the development process than in traditional programming life cycles. It is imperative that a process be established which enables maintainers to track and recreate earlier versions of the expert system.

An incremental development process also tends to be subject to tinkering; constantly modifying various components of the program. The danger in this approach is the possibility of changing software without any management of the process. In such an instance, resources are being applied piecemeal with little value-added to the product.

The ease with which changes can be made in today's expert system shells is yet another challenge to configuration management.

Rapid prototyping need not be uncontrolled prototyping. The CM process does not have to restrict creativity or the development cycle. In fact, as Kolkhorst (1994) points out, the CM process can be prototyped right alongside the expert system. This allows for the establishment of a flexible, evolutionary CM process that can grow to meet the needs of the organization.

4. Frequency and Sources of Change

Hicks (1990) argues expert systems are among the most difficult software to maintain because:

In addition to the problems expected with large systems, the knowledge maintained is normally more dynamic than traditional data processing applications, is not common or well documented, may be distributed, and must be acquired and tested incrementally.

In addition to the high rate of change common to expert system development, the existence of several sources of change poses additional challenges. The coordination and collection of changes from various sources must be an integral part of any expert system CM process. For example, in the case of the MK 92 MAES project, inputs from the fleet, NSW engineers, ORDALTs, as well as changes made by manufacturers of the fire control system must all be accounted for and implemented. Additionally, the CM process designed for the MK 92 MAES must incorporate changes made to technical manuals, and Planned Maintenance System (PMS) procedures.

5. Expert System Development Environment

The expert system development environment can also add difficulty to the implementation of CM. Programmers could be geographically distributed and so is the knowledge required for developing the system. The communication problems created by this situation complicates the CM of the knowledge base and the implementation of the

expert system. The CM of the knowledge base and implementation of the expert system will be elaborated upon in a later section.

Consider the case of the MK92 MAES project. As noted in chapter II, the MK92 MAES represents a cooperation between NSWC, Port Hueneme and the Naval Postgraduate School. Domain experts and technical representatives are provided by NSWC PHD. Implementation personnel are made up of faculty and graduate students from NPS. The physical distance between NSWC-PHD and NPS and the work environment of both organizations make communication difficult. Furthermore, the academic environment of NPS limits the opportunity for face-to-face visits between graduate students, who have classes to attend in addition to their participation in the project, and domain experts, who are working on resolving problems for the fleet.

Additionally, due to the inherently autonomous nature of an academic environment, each student works on his/her own particular aspect of the project as part of his Master's Thesis. Different students work on different modules, to accomplish different tasks. Each needs to coordinate his efforts with fellow students.

The turnover rate of project team members also highlights the need for configuration management. As programmers transfer to other assignments, they take with them the corporate knowledge they have acquired while working as part of the development team. For instance, the MK92 MAES project experiences a high turnover rate of the student developers. Rarely, if ever, is a student involved with the project for longer than 12-15 months. While not affecting the strategic focus provided by the faculty, the loss of corporate knowledge at the implementation level has underscored the need for an effective CM process.

The work environment of the development team, coupled with the rate at which student programmers transfer necessitates a seamless, easy to understand CM program.

B. CONFIGURATION MANAGEMENT OF THE KNOWLEDGE BASE

While CM is applied to expert system development much in the same way as it would be applied to any prototyping activity, the main difference between the CM of an expert system development and that of a traditional program lies in the application of CM to an expert system's knowledge base.

Although CM is applied to the expert system implementation, a CM process that parallels that of the expert system's process should be established for domain knowledge. This ensures the integrity of the knowledge is maintained independent of the method of its implementation. A poor implementation of accurate knowledge can be reworked; however, if the knowledge has poor integrity, there is no way to establish the expert system's accuracy. By applying CM to expert knowledge, its reusability and portability is enhanced.

This sub-section discusses in greater detail, the configuration management of an expert system's knowledge base.

1. Characteristics of the Knowledge Base

Hicks (1990) identifies three factors which impinge upon the maintenance of knowledge; volatility, expanding functional scope, and system size/complexity. Since maintenance is the implementation of changes to the expert system, these factors influence the need for CM of the knowledge base.

a. Volatility

Volatility refers to the frequency of changes to the knowledge base. Hicks (1990) considers this to be the "single largest factor in maintenance." Some expert systems contain a knowledge base which has thousands of changes each year. In one example, Hicks (1990) describes an expert system in which half the code is rewritten each year. In another, changes would only be made if a change was made to the organization's infrastructure.

The more volatile the changes experienced by an expert system, the greater the maintenance burden. A highly volatile knowledge base requires a highly responsive process for identifying, controlling and implementing changes.

b. Expanding Functional Scope

Expanding functional scope refers to the growth of the rule or knowledge base in terms of functionality. As an expert system proves its value, there is a tendency to increase its functional capabilities (Hicks, 1990). The addition of this functionality into a pre-existing knowledge base presents a CM challenge. If the growth of the knowledge base is not a controlled process, resources may be wasted, and maintainability could be lost. A configuration management process needs to be implemented that will control what enhancements are introduced, in what priority, and with what resources.

c. System Size/Complexity

The complexity and size of an expert system will also affect its maintainability. "Large software programs are notorious for being difficult to maintain and expand (Hick, 1990)." Furthermore, as the expert system's knowledge base grows, the complexity of the relationships and dependencies could make the knowledge base unmaintainable. A configuration management process, in conjunction with sound development strategies, could ensure changes to a system are documented, and traceability is maintained.

2. Distributed Knowledge

Knowledge required to develop an expert system could be obtained from several sources that are physically distributed. The sources of knowledge can include technical manuals, system documentation, domain experts, as well as other sources. To coordinate the acquisition, representation, and implementation of such knowledge, it becomes necessary to establish control over the process. Configuration management can provide this support.

3. Knowledge Representation Scheme

Algorithms are generally easier to change and maintain than the heuristic knowledge of expert systems. The relationship between various portions of an algorithm are generally explicit, and can be easily tested before implementation. Likewise, the expected outcome of a change is more easily predicted and can be compared to the actual outcome of the implementation.

The relationships encapsulated in a knowledge representation scheme may not be as obvious. As rule sets grow large, the complex inter-relationships of a knowledge base can be difficult to maintain. This is compounded by the selection of a representation scheme which does not depict a domain expert's knowledge in an intuitive and logical manner. Easy to understand knowledge representation schemes enable maintainers to quickly identify the causes of problems and assess the impact of proposed changes, and therefore simplifies the CM process.

Ideally, a representation scheme will promote traceability from the implementation to the domain expert's knowledge. The closer the match between the way the domain expert approaches the problem and how the knowledge is represented and implemented, the easier the task of managing changes will be.

4. Configuration Identification of Knowledge

To apply configuration management to a knowledge base, it is necessary to establish what will be managed, or in other words it is important to define, what is a configuration item. To accomplish this, one needs to identify how knowledge is captured, and at what point it needs to be controlled.

As a rule of thumb, knowledge should be brought under configuration management when there is a possibility it could be incorporated into the expert system's knowledge base. The degree to which it is to be controlled may vary, but some level of CM should be established.

As stated in Chapter II, knowledge is traditionally elicited through interviews or questionnaires. Using this method of knowledge acquisition, an expert's knowledge is

elicited by a member of the development team. Once captured, it should be placed under CM.

As with traditional software, the degree to which a knowledge configuration item is controlled is determined by its category. However, unlike the categorization of software, knowledge categories may contain different formats. This includes audio taped interviews, responses to questionnaires, drawings, technical manuals, etc.

The following is a recommendation for cataloguing domain knowledge.

a. Working knowledge

Working knowledge is that knowledge which the domain expert is developing as his/her input to the knowledge acquisition process, but has not been provided to the development team. It receives the lowest level of CM, and is synonymous to the contents of a programmer's working library. With working knowledge, the primary concern is maintaining separation between that knowledge which the domain expert has turned in to the development team for representation and that which is still work-in-progress. This could be done by simply keeping the two types in separate file folders, or if using an automated tool, separate sub-directories.

Working knowledge represents a classification which will generally not be seen when using the interview process. The distinction is whether or not the knowledge has been provided to the implementation team. For example, knowledge acquisition for the MK92 MAES was accomplished using diagnostic trees drawn by the domain expert. Until provided to the development team at NPS, there is no possibility of it being incorporated into the expert system; therefore, very little CM is required. Generally all that is necessary is the use of naming conventions, and minimal access control to prevent its inadvertent inclusion in knowledge to be represented.

b. Captured Knowledge

Captured knowledge is knowledge that has been elicited by a project team member, is in the development team's possession, and could potentially be incorporated in the expert system. Though in a form which does not reflect the chosen representation

paradigm of the expert system, some measure of control should be in place because the knowledge has come under the development team's control.

Some examples of captured knowledge include interview tapes, completed questionnaires, drawings, technical manuals, and diagnostic trees. The reader should note that diagnostic trees may fall under the category of represented knowledge. This is particularly true if a procedural network representation paradigm is to be used for implementation. The identifying feature of captured knowledge is that it has not been represented using the selected expert system paradigm.

c. Represented Knowledge

Represented knowledge is that knowledge which has been captured and represented using the chosen knowledge representation paradigm (e.g., frames, rules, procedure networks, semantic networks, etc.) Represented knowledge is knowledge which has not yet been selected to be incorporated in the expert system's knowledge base. This may be due to the fact it is still undergoing testing or is outside the boundaries of the expert system's domain. Because it is only a step away from implementation, it will be subject to a higher degree of CM.

d. Product Knowledge

Product knowledge is that knowledge which has been identified for incorporation into the expert system's knowledge base. It should be under the most stringent level of knowledge base CM.

5. Baselining Knowledge

Similar to traditional software development, knowledge should be baselined. By baselining and applying CM to the expert's knowledge independent of the expert system software, the domain expert's knowledge is maintained separately from its implementation. This ensures the knowledge can be re-implemented using a different expert system shell, or represented using a different paradigm if necessary.

The knowledge contained in a particular knowledge baseline forms the knowledge base of the corresponding baseline of the expert system. For example, the expert system's product baseline should implement the same knowledge as is represented in the product knowledge baseline. This allows for easier maintenance and traceability.

The author recommends the following baselines for domain expert knowledge:

a. Functional Knowledge Baseline

This contains the most recent product knowledge baseline. This baseline represents the functionality of the expert system, and therefore is a key component in the expert system's functional baseline.

b. Developmental Knowledge Configuration

This knowledge is the most current version of the domain expert's knowledge. It is in this baseline that all changes are made to the domain knowledge. The expert system's developmental configuration is traced to this baseline.

c. Product Knowledge Baseline

This baseline represents the domain expert's knowledge which has been implemented in the current release of the expert system. Upon establishing a particular product baseline for the expert system, the knowledge represented in the product knowledge baseline is used to describe the functionality of the expert system. For this reason it is used to establish the expert system's functional baseline.

6. Assessing the Impact of Changes to Knowledge

One of the primary difficulties in making changes to an expert systems knowledge base is assessing the impact of proposed changes. Expert systems are typically developed to reduce an organizations dependence upon domain experts. Often this is the result of a decreasing a knowledge pool, or the anticipated loss of a particularly knowledgeable expert (Prerau, 1990) (Walters & Nielsen, 1988).

The development of an expert system suggests the capturing of knowledge an expert has that cannot be obtained easily from algorithmic sources (Walters & Nielsen,

1988). The effect of proposed changes may not be immediately obvious to the developer/maintainer, thereby complicating its impact assessment. For this reason, the domain expert or other technically proficient personnel must be involved in the configuration management process. Where it is relatively easy for the programmer to determine how long it will take to implement a particular change, the potential "ripple effect" of the change may not be known. This is true to a much larger extent in expert systems than in traditional programs.

By definition, expert systems implement the heuristics in a domain expert's thought processes to arrive at a solution in a constrained problem domain (Prerau, 1990). Because the expert system is a representation of the thought processes of the domain expert, it is highly desirable to ensure they remain a key participant in the maintenance process. Since this may not be possible, a detailed change control and configuration library are paramount to ensure the integrity of the development process. By establishing CM, technical experts in a particular problem domain are provided with a change history of the knowledge base and can gain insight as to the thought process of the expert whose knowledge has been captured. This may assist them in accomplishing maintenance functions more efficiently.

7. Controlling Changes to the Knowledge Base

As noted by Bielawski and Lewand, (1988) expert systems are rarely complete. The knowledge base will always evolve, requiring a process which will control changes so they are made in a logical and economical manner. This process in CM is configuration change control.

a. What Influences the Change Control Process?

In addition to the volatility, functional scope, and complexity of the knowledge base, other factors influence the change control process. The method of knowledge acquisition will impact the frequency with which changes are made to the knowledge base. A highly iterative process such as interviewing will result in more frequent changes. A process in which the domain expert manipulates the knowledge as

working knowledge, refining it until he is satisfied before giving it to the development team will reduce the number of changes developers must make.

b. When Should Changes Be Made to the Knowledge Base?

Bielawski and Lewand (1988) recommend four instances in which maintenance should be performed on expert systems. Extending their recommendations to the knowledge base, changes should be made when:

- bugs or inaccuracies found in the knowledge base impair the performance of the expert system significantly
- new knowledge is necessary to update the expert system
- the thought process captured no longer is capable of solving the problem
- the knowledge must be changed to accommodate a new representation paradigm

c. To What Knowledge Category Should Change Control Extend?

Change control could conceivably extend to every time a knowledge engineer interviews a domain expert to capture knowledge. However, this would be impractical. To reduce the overhead associated with configuration change control of the knowledge base, it should only be extended to that knowledge which has been formatted in the knowledge representation scheme chosen for the expert system implementation.

For example, if one were to conduct an interview using an audio tape, some degree of control would need to be placed on the audio tape. Perhaps a control which would be designed to prevent the accidental overwriting, loss, or erasure of the knowledge elicited. However, once the knowledge from an interview has been represented in a manner which will facilitate its coding, greater change control is necessary. This is due to the fact, the representation scheme could inadvertently be encoded without proper testing, verification, and validation of the knowledge.

Not every change to knowledge under the control of the development team requires extensive change control. Only those which are represented for encoding in the expert system need be controlled. As a general rule, CM should be applied to knowledge

when it is first considered to be a candidate for implementation and represented in the selected paradigm.

Chapter II introduced the knowledge acquisition methodology used by the MK92 MAES project team. The traditional knowledge acquisition process, is one in which a domain expert's knowledge is elicited through interviews, documented, then tested in an iterative manner. In this methodology, the expert represented his own knowledge using diagnostic trees, thus eliminating the need for knowledge engineers. Just as the programmer has a working directory for code "in progress," the domain expert under this approach is given the same privilege. Where a domain expert's knowledge acquired using traditional knowledge acquisition techniques would be brought under CM when first documented. The knowledge acquisition technique of the MK92 MAES project does not require the domain expert to do so until turned over to the development team for coding.

This novel approach to knowledge acquisition represents a change management challenge. The configuration management process must include the domain expert. Because the domain expert may not be well versed in software engineering disciplines, it is important to educate them as to the purpose and importance of configuration management. Policies and procedures must take into account the need to apply CM to the domain expert's role in the expert system development process.

C. CONFIGURATION MANAGEMENT OF EXPERT SYSTEM SOFTWARE IMPLEMENTATION

The configuration management of expert system software is identical to that of traditional software applications. Identification, change control, status accounting and audits must all be undertaken in a manner which provides the degree of control necessary for maintaining product integrity. However, the challenge of managing expert system software extends from the technology inherent in many expert system shells.

Expert system development tools may not lend themselves easily to tracking changes to the expert system. Designers of expert system shells focus their efforts on the

functionality associated with the knowledge representation and coding paradigm's they have selected for their tool. Very little, if any, attention is given to facilities which would aid in the application of sound software engineering practices to the expert system development process. (Prerau, 1990)

This situation applied to the expert system shell selected for use in development of the MK92 MAES. As discussed in Chapter II, the expert system shell, Adept, is a visual programming tool. Whereas some visual development tools such as Visual Basic, allow software under development to be saved in a text format, Adept stores developed code in a proprietary binary format. CM tools that generate detailed reports on text based files can still be used for change control and version control, however their utility as a tool for tracking revisions and identifying the differences between two binary files is limited. As a result, the change control, status accounting and configuration auditing tasks of CM are made more cumbersome. Instead of applying an automated CM tool to track changes to Adept code, they must be accounted for using a manual approach.

Visual programming languages, such as SoftSell's Adept, present new challenges to CM. Pre-defined visual representations, such as the nodes and arcs in Adept, represent code at a higher level of abstraction than third generation languages (3GLs) such as Ada, or C. Therefore, if one wishes to apply CM to the entire coding scheme of a program, CM must be applied not only to the visual representation generated by the developer of an application, but also to the underlying code defining the functionality of the visual node or object.

For example, Adept gives a developer the ability to build customized nodes through the use of its script language, NodeTalk and to establish relationships between these nodes through connections called arcs. The need is apparent for the CM of developer-generated custom nodes as well as the arcs between them. As changes are made by the development team, one would want to control changes through the CM process.

This concept of applying CM extends to the pre-defined nodes provided by the vendor. Consider the instance, where the vendor, through the evolution of their product makes a change to the underlying code which is represented by a particular node. To the user, the pre-defined node will not appear to have changed, even though the code implementing the pre-defined node's functionality may have changed dramatically. When integrated with existing modules, there is no way to be certain the redesigned node will not interact negatively with those defined by the user. Unless this process is managed, changes made by the vendor could seriously impact the development team's product. (Kolkhorst, 1994)

The application of CM to pre-defined nodes, as noted in chapter III, need not be done by the developer of the expert system. It will, in all likelihood be managed by the vendor. The lengths to which the expert system project team should go to ensure CM is being applied to the vendor defined nodes will depend largely on the "criticality" of the expert system under construction. When possible, the developer should attempt to obtain the code for altered pre-defined visual objects, in order to evaluate the impact of any changes. (Kolkhorst, 1994)

V. CONFIGURATION MANAGEMENT TOOLS

The trend toward the establishment of Configuration Management (CM) as an integral part of government software development projects and CM's increased use in the corporate sector has created a demand for automated approaches in its implementation. Where there was once only CM tools for mainframe based application development, today there are CM development tools for Windows, UNIX and other platforms with greater capabilities and compatibility than before.

The main purpose of using a configuration management tool is to relieve the development staff of the burden of implementing configuration management. The tasks of configuration identification, change control, status accounting, and auditing would be tedious on a small project and virtually unmanageable on larger efforts without some automated support to manage key aspects.

This chapter examines the desirable features of an automated configuration management tool. In addition, it presents a framework of the main features a user should consider when evaluating a CM tool for a particular project is undertaken. Finally, the framework is applied to the evaluation of the features and subsequent selection of a configuration management tool for use in the MK 92 MAES project.

A. CRITERIA FOR SELECTING CONFIGURATION MANAGEMENT TOOLS

In general, there are two facets to evaluating a generic configuration management tool for use in a software project. The first relates directly to the features of the tool. The second deals with the organizational, project management, and human factors issues related to the use of the tool. The following subsection explores these two facets in detail.

1. Configuration Management Tool Features

The term features, in this instance refers to the functional characteristics of a configuration management tool. A good CM tool should support all tasks of configuration management. As a minimum, a configuration management tool should be able to assist the project management and developers in identifying configuration items;

providing a change control feature; supporting a mechanism for status accounting; and establishing and maintaining an auditing capability (Wreden, 1994).

a. Change Control

Change control features ensure changes made to one version, cannot be incorporated in another without some explicit action on the part of the programmer. Change control features include check-in/check-out and lock-out functions, as well as the ability to identify the changes made to a particular version.

b. Version Control

Version control enables the developer to store, maintain, and recreate various versions of a product. Given the possibility many versions of a program could exist at any one time, this capability is crucial in debugging problems identified by customers. Version control ensures a version of software can be recreated any time it is desired (STSC, 1994).

c. Reporting/Query Capability

A reporting/query facility enables development personnel to generate reports which can be used for project management (STSC, 1994). Reports denoting differences between two or more versions of software, change status, current configuration items under CM are typical reports of CM tools. In addition, a reporting facility is fundamental in automating configuration management status accounting.

d. Library/Repository

All CM tools have a library/repository. A library acts as a storage facility for software configuration items. Generally, a programmer will check an item out of the library for coding or testing. When completed, the item will be checked back in. In addition to code, some CM tool's store documentation, test routines, graphics, or other configuration items.

e. Release Management

Release management features assist in identifying what items are on a particular version through the automatic generation of version description documentation, electronic media labeling, etc. (STSC, 1994)

f. Compatibility

Compatibility describes the extent to which a CM tool can communicate and share information with other tools. Some CM tools have the capability of integrating with other products such as problem tracking tools, build tools, document management, report generators, as well as other Computer Assisted Software Engineering (CASE) tools. Other tools have been ported across multiple platforms, allowing developers working on PCs or UNIX workstations to use the same CM library. Compatibility with other development software allows for flexibility and expandability should the current effort's CM needs change.

g. Build Support

Build support is a feature which enables a CM tool to not only track source code, but also link and compile it. Some tools will only compile those portions of code which have changed. (STSC, 1994) Build support facilitates the integration of the tool with the development process. The greater a CM tool's role in the day to day software development environment, the more likely its use will be adopted.

h. Team Support

Team support describes the ability of a tool to support distributed development (STSC, 1994). As hardware trends have shifted from mainframe technology to the use of local area networks (LANs), software development has evolved into an effort in which many programmers are likely to be making changes to a configuration item at any one time. A CM tool with good team support capabilities will have a mechanism for locking a configuration item to minimize the possibility of simultaneously attempting to make changes to different versions of the same module.

i. Ability to Customize Features

The ability to customize the interface, life cycle model, or other aspect of a CM tool is a highly desirable feature. As a project's needs change, or as the CM process is improved, there will be a desire to alter various aspects of the life cycle model or other aspects of the configuration library. By using a tool that can be extensively tailored, flexibility and expandability are increased.

2. Additional Selection Criteria

Aside from the configuration management features of the tool, there are other considerations which must be taken into account. These include human factors issues, managerial issues, and product issues not specifically identified as configuration management features. The following is a listing of some of these criteria, some of which have been identified by the Software Engineering Institute (SEI) as points for consideration when selecting a CM tool (STSC, 1994).

a. Cost

The term cost refers not only to the price of the CM tool, but to the cost of maintaining it. Maintenance costs include time, personnel, and money.

b. Ease of Incorporation Into a Project Life Cycle

Ease of incorporation into a product life cycle refers to the effort required to integrate a CM tool into the project's life cycle. The amount of data conversion required to implement the CM tool and the ability to import information from existing databases are two considerations that determine the ease with which a tool may be incorporated. An additional feature is the ability to create life cycle models for CM tool archives. A tool which can be more easily folded into the project team's efforts will minimize the impact of the tool's introduction to the organization.

c. Ease of Use

If a tool is not simple to use, it will not be used. Steep learning curves and difficulties experienced by the end user of the product will ensure CM will not be adopted.

User interface, system documentation, help facilities and tutorials are features that enhance a product's ease of use.

d. Security

Security is a desirable feature in a CM tool for two important reasons. First, there is a need to control access to product libraries. Developers should not have unrestricted access to all aspects of the configuration process. For instance, the configuration manager might wish to control who has authorization to migrate changes from the development or testing libraries to the product library. In this way, product integrity can be maintained. Secondly, there is always the danger of someone accidentally overwriting good code with untested code. A security structure establishing access controls could prevent such occurrences. Finally, a security feature would prevent unauthorized copying, thereby ensuring untested modules are not sent to customers inadvertently.

e. Power

Power refers to the extent to which one command have an effect throughout the CM tool and the project's CM library. Tools with recursive check-in and check-out features, change packaging utilities, and package migration capability simplify the jobs of the configuration manager and CM library administrator. (STSC, 1994)

f. Robustness

Robustness refers to a the reliability of the software. Such factors as how the tool performs under failure conditions should be considered. Whether or not checked-out items remain locked if there is a system failure is another consideration for a CM tool. Finally, features which allow the configuration library administrator to reset tool settings are indicative of a robust configuration management tool.

g. Scalability

Scalability addresses the issues surrounding the size of the project to which the tool is going to be applied. Is the tool the right size for the project? How many items

can be managed at any one time? How extensive is the management structure? What kind of life cycles can it support? Can it expand to meet our future needs, regardless of scale? One wants to select a tool which will not only meet today's needs, but can satisfy the demands of tomorrow.

h. Quality of Commercial Support

Many of the risks associated with the above factors can be minimized by quality commercial support. If the vendor is responsive and has a strong technical support infrastructure, their assistance can be relied upon to overcome difficulties. What is the vendor's reputation for support? Are they helpful; dedicated to resolving conflicts and bugs found in the particular version of the CM tool, or are they elusive? How strong is the maintenance agreement?

The availability of training may also be a factor for evaluation (IEEE, 1992). If there are no personnel skilled in CM, the project team may desire training. Availability of training is one indicator of a vendor's support structure. If resources are going to be applied to implementing a CM process, one needs to be certain the vendor of the selected CM tool will be there when needed.

The degree to which, the CM product is supported by other CASE tool vendors should also be taken into account (IEEE, 1992). A CM tool not supported by other vendors indicates to the potential buyer one of two things; either the CM tool's vendor is unwilling to release code which will allow others to coordinate integration efforts, or it sheds light on the acceptance of the CM tool by the marketplace. A CM tool which no other vendor is building interfaces for is not as favorable as one which can be integrated with other development products .

i. Project Specific Features

The list of features and selection criteria presented here is not exhaustive. It is intended to identify some of the considerations one must make when selecting a CM tool. It will be necessary to identify additional features that are required to satisfy the needs of a project due to some particular aspect of its development environment. For

instance, a visual programming environment may require features that enable a tool to track changes made to its code. The features one might wish to consider will differ between projects. This is true with the MK92 MAES. Section C, subsection 3, presents selection criteria for the MK92 MAES that were not mentioned in this section.

3. Further Reading

For further reading on CM tool selection, and specific CM products, the reader is referred to the Software Technology Support Center's report on configuration management technology (STSC, 1994) and IEEE's Recommended practice in evaluating and selecting a CASE tool (IEEE, 1992).

B. AN EVALUATION METHODOLOGY FOR CM TOOLS

This subsection presents the evaluation process for the selection of a CM tool. This methodology assumes the person or team making a tool selection has a general understanding of configuration management concepts. It is adapted from IEEE's recommended practice on CASE tools selection (IEEE, 1992). The First three phases can be conducted in parallel, while the last three should be conducted in the order presented.

1. Conduct a Literature Review

The first step is to conduct a literature review. The purpose of the literature review is to acquaint the selection team with the desired features of CM tools and the state of CM tool technology. The extent to which a literature review is necessary will depend upon the experience level of the selection team.

2. Identify Constraints

Constraints act as the decision boundaries. An effort should be made to identify those variables which will pose restrictions on the tools which can be selected. Such considerations as hardware limitations, operating systems, and budgetary restrictions must be identified.

3. Identify Evaluation Criteria

This step includes the establishment of objective and subjective criteria that will be used in the comparison of competing CM tools. IEEE (1992) recommends applying weights to the evaluation criterion. For example, if cost is the primary goal, then more weight should be assigned to it than to other factors. One way in which this can be done is to list the criteria in order of importance.

4. Identify Candidate Configuration Management Tools

At this point, the selection team should identify CM tools which claim to meet the minimum requirements of the evaluation criteria. The list of candidate tools should be compiled from available literature, market surveys, vendors, and other sources. One should then attempt to obtain evaluation copies from vendors and literature evaluating the performance of the candidate tools, when available. (IEEE, 1992)

5. Evaluate Candidate Configuration Management Tools

If demonstration copies are available, compare one tool against another using a pilot project that is representative of the target project. If establishing a pilot project is not possible, compare the individual features of each tool using the evaluation criteria identified in Subsection 3.

6. Make a Selection

Once the evaluation is completed, the selection team should select that tool which maximizes the evaluation criteria. It should be noted that there is no single tool which will encompass all features desired in a CM tool. The goal is to select that tool which best incorporates the features which are required for the development project on hand.

The key to successful tool selection is to analyze the needs of the program. Both current and future development efforts must be taken into account. CM is conducted for the entire life cycle, and the needs analysis must take this fact into account. One should not assume that because the organization is not using distributed development today, it

won't require it later. A tool should be chosen that can meet the present and future needs of the project.

C. APPLICATION OF EVALUATION METHODOLOGY TO THE MK92 MAES

This subsection applies the previously developed methodology to the selection of a CM tool for the MK92 MAES project.

1. Conduct the Literature Review

Articles from CD ROM archives, trade magazines, STSC (1994) technical surveys, and Internet frequently asked questions (FAQ) were reviewed. This literature review familiarized the author with the features of CM tools. This enabled the identification of constraints and evaluation of a CM tool for the MK92 MAES.

2. Identify Constraints

The first constraint was that the tool had to run in a Windows 3.1 environment to provide as seamless an environment as possible for the developer. While there are many CM tools available, only a small number of them run under Microsoft Windows (Eaton, 1994).

As is the case for many small projects, a second constraint was a budgetary one. A dollar ceiling of \$500 was established as the maximum that could be afforded. As a result, our search range was limited to the following products in alphabetical order:

- CCC/Manager by Softool Corp. Softool provided a demonstration version for evaluation. The features, comparisons, and findings of this evaluation are presented in the following subsection.
- PVCS Version Manager by Intersolv, Inc. This tool was purchased at the educational price of \$100 and is also evaluated in the following subsection.
- SourceSafe by Microsoft. Formerly a product of One Tree Software, SourceSafe was purchased by Microsoft late in 1994. Although evaluation copies were provided when owned by SourceSafe, Microsoft no longer supports this policy. Though SourceSafe has a good reputation, the lack of a demonstration copy prevented its evaluation.

As a result, SourceSafe was removed from further consideration.

3. Identify Tool Evaluation Criteria

The CM tool selected had to support the four tasks of configuration management identified in Chapter III; configuration identification, change control, status accounting and auditing. In addition the tools were compared on the basis of the following features.

- **Cost.** Although identified as a constraint, it is included here as an evaluation criteria.
- **Security.** Security includes access controls, user privileges, password capability, and encryption support.
- **Change Control.** Change control refers to the quality of those features that support change control.
- **Check-in/check-out functions.**
- **Lock-out function.** Can the tool lock a file when it is checked-out to prevent another programmer from accessing it?
- **Ability to track changes to Adept procedures and binary files.**
- **Version Control.**
- **Ease of Incorporation in the Expert System Life Cycle.** This refers to the ability for a tool to model, through the establishment of archival directories, the MK92 MAES' development life cycle.
- **Ease of Use.** Described in Section A.
- **Interface.** This term refers to the quality of the user interface.
- **Ease of installation.** This term refers to the difficulty involved with implementing the CM tool.
- **Ease with which users can learn to use the tool.** Self-explanatory.
- **Quality of the system documentation.** Self-explanatory.
- **Quality of on-line help.** Self-explanatory.
- **Availability and Quality of tutorials.** Self-explanatory.
- **Quality of CM Reporting Functions.** Self-explanatory.
- **Team Support.** Described in Section A
- **Customizability.** Customizability was explained in Section A

- Interoperability with Other Tools. Self-explanatory.
- Build Support. Described in Section A
- Release Management Functions. Described in Section A.
- Performance. Performance was measured by evaluating the time (in seconds) it took to check-out a representative MK 92 MAES file from each tool's product support library.
- Overall suitability of features to the MK92 MAES CM process. General Impression of the Tool's suitability to the MK92 MAES project.

4. Identify Candidate CM Tools For Further Evaluation

Two tools met the minimum criteria to be considered for further evaluation. CCC/Manager and PVCS Version Manager were selected for further evaluation. Subsection 5 is a brief description of CCC/Manager. Subsection 6 is a description of PVCS Version Manager.

5. CCC/Manager

CCC/Manager is a configuration management tool developed by Softool Corporation. The version examined is version 2.2.2. Acting as a database, CCC/Manager controls the changes to a project, provides access control and reporting features. The following is a summary of the primary features of CCC/Manager. Files from the MK92 MAES are used as examples to demonstrate the capabilities of CCC/Manager. Descriptions of CCC/Manager's functionality are adapted from "CCC/Manager Primer" (Softool Corporation, 1994a).

a. Cost

CCC/Manager costs \$495 per user. Maintenance is \$85 per year as of the 1994 CCC/Manager price list.

b. Interface

The interface for CCC/Manager is a windows based GUI. The primary operations within CCC/Manager are accomplished using three primary windows. The Configurations window displays a project's configuration management libraries. Each

library in CCC/Manager is the CM repository for a particular lifecycle. The project's configuration library administrator creates the configurations for a particular project. Configuration items are migrated between the CCC/Manager configurations as the software progresses through the development lifecycle. Figure 5-1 is an example of the Configurations window for the MK92 MAES.

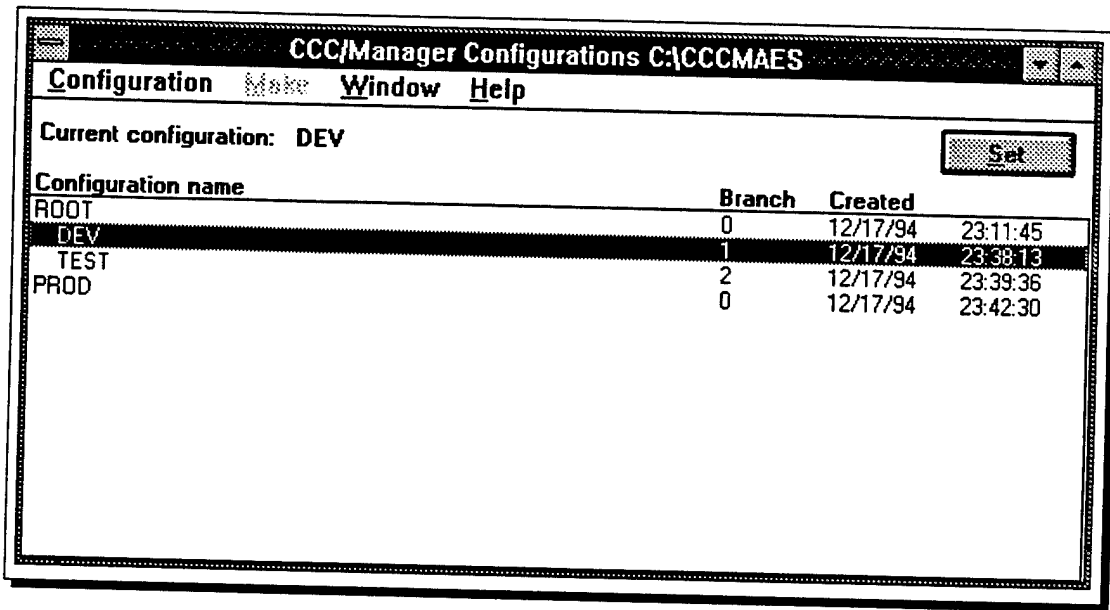


Figure 5-1. CCC/Manager Configurations window.

When files are checked into CCC/Manager, they are referred to as items. These configuration items are viewed and manipulated in the Item Functions window. Figure 5-2 is an example of the Item Functions window for the MK92 MAES. The Item functions window contains both a directory tree and a list of the files contained in the directories, just as would be found if one were to open the File Functions window. In addition, the Item Functions window displays the initial and current version numbers of each item checked into the library. In addition to the version numbers, the window displays who has locked out a file. The Item Functions window is the screen from which all configuration library manipulation functions are initiated.

The File Functions window is where the PC files are identified and manipulated in CCC/Manager. In addition to file functions this window provides the

facilities for establishing user settings and access controls. Additionally, the File Functions window contains the database administrator functions for use by the configuration library administrator. Figure 5-3 is an example of the File Functions window for the MK92 MAES.

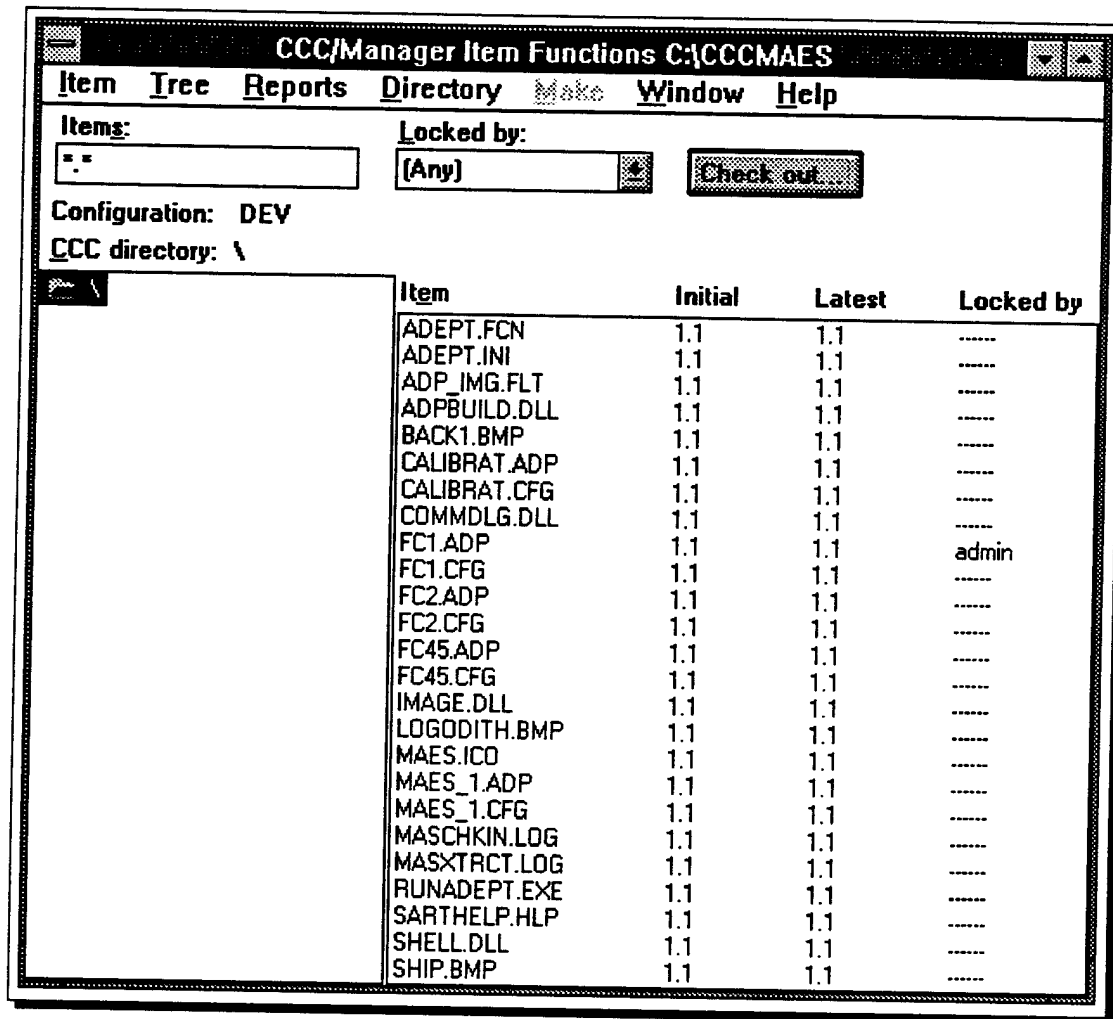


Figure 5-2. The Item Functions window for the MK92 MAES.

c. Access Control

Access controls limit who has access to what within the configuration library. CCC/Manager provides access control through several methods. All of these controls are established by the configuration library administrator (CLA) through the DBA

functions found on the File Functions window. The first access control established in CCC/Manager is the use of passwords at a log-in screen. The user cannot gain access to any portion of the CM library without a password. (Softool Corporation, 1994b)

File	Size	Date	Time	Attrib
adept.fcn	204800	1/5/94	19:47:46	ar
adept.ini	564	1/27/94	8:51:52	ar
adp_img.flr	14717	12/31/92	16:30:12	ar
adpbuid.dll	243584	1/5/94	19:47:46	ar
back1.bmp	405622	12/14/93	15:16:26	ar
calibrat.adp	1601536	8/30/94	18:54:20	ar
calibrat.cfg	119	2/15/94	16:31:04	ar
commdlg.dll	89248	1/17/94	16:11:52	ar
fc1.adp	2695168	1/13/95	16:19:52	ar
fc1.cfg	119	8/11/94	12:45:56	ar
fc2.adp	3740672	8/24/94	17:15:42	ar
fc2.cfg	116	5/26/94	9:59:16	ar
fc45.adp	2396160	8/24/94	0:18:50	ar
fc45.cfg	119	8/24/94	0:18:28	ar
image.dll	107917	10/1/92	17:35:00	ar
logodith.bmp	9782	9/18/91	9:23:54	ar
maes.ico	766	5/21/94	17:26:12	ar
maes_1.adp	57344	8/30/94	18:47:00	ar
maes_1.cfg	84	6/22/94	17:59:02	ar
maschkin.log	1720	1/9/95	20:22:00	ar
masxtrot.log	2238	1/9/95	21:08:28	ar
runadept.exe	393696	1/5/94	19:47:46	ar
sarhelp.hlp	75491	6/29/92	18:13:14	ar
shell.dll	41600	3/10/92	3:10:00	ar
ship.bmp	192018	9/17/91	14:34:50	ar

Figure 5-3. File Functions window for the MK92 MAES.

Access controls are further delineated through the assignment of users to a particular user group. There are three primary user groups; user, manager, and administrator. Each has a default level of access that can be tailored to the project's needs by the CLA. In addition, the CLA can create new User Groups which can be assigned access, thereby allowing for a greater degree of customizing the system.

In addition to assigning permissions for access to CM functions, the CLA may also establish configuration restricted control. Through configuration access control, not only are users access to a specific configuration, the capabilities they have within a particular configuration can be restricted as well. A user group may be assigned read only, limited, or full access to a particular configuration. Figure 5-4 is an example of a screen in which the CLA can assign access controls.

d. Life Cycle Modeling

CCC/Manager, through the use of the Configurations window, provides for the establishment of an automated CM process which models a project's software life cycle. When the CM library is first created, the CLA establishes the libraries which the CM program will use. It is from these libraries that users check out, change and migrate files, generate reports, and promote files for release as products. By mapping the phases of the life cycle to CCC/Manager configurations, development personnel are able to work

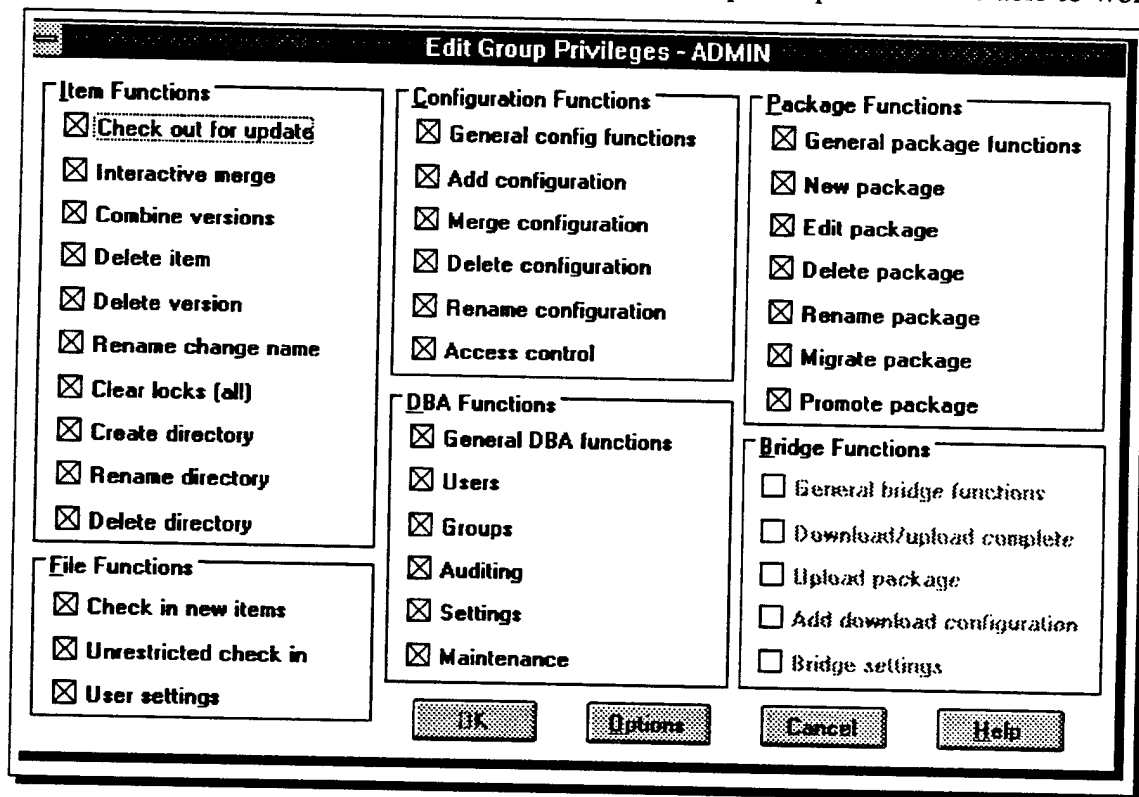


Figure 5-4 An example of group privileges for the MK 92 MAES.

within a CM framework which matches as closely as possible, the development environment of the project.

e. Check-in/Check-out

In addition to the ability to provide fundamental check-in/check-out capabilities, CCC can conduct recursive check-in/check-out. Recursive check-in/check-out refers to the ability to check-out files as a group for work which may require more than a single file. This capability prevents the developer from having to check-out each file one at a time.

In addition to check-in/check-out, CCC/Manager has the capability to lock-out files which have been checked out for maintenance. This prevents two users from simultaneously updating a file, thereby overwriting the changes each made to the file.

f. Change Control

When one desires to make a change, one first checks out the file from the Item Functions window to the programmer's working directory. As figure 5-2 demonstrates, the file is then locked, thus preventing access by others. The user then makes changes to the file as desired. Upon completion, the developer checks the file back in. Files are checked in through the File Functions window.

Figure 5-5 is an example of the check-in dialogue box. In order to complete the check-in process, the user must give the change a title and brief description. This encourages the documentation of the change process.

Several changes may be grouped together to form packages. These change packages enable the developer to manipulate groups of changes more easily. This procedure speeds up the process of migrating and promoting changes.

g. Version Control

CCC/Manager provides for the capability to reconstruct any version that is part of the configuration library. It does this for text files by storing the changes to a

particular file as "deltas." These "deltas" are then applied to the baseline to reconstruct any particular version of the code desired.

In the case of binary files, there is no feasible way to identify specific changes; therefore, "deltas" cannot be created. In the case of binary files, CCC/Manager stores the entire file. When an older version is desired, it is retrieved from the configuration library.

Check In - DEV

From
PC directory: c:\adept\maes
Files: *.*

To
CCC directory: \

Change name: STR0001
Comment: This change is to FC1. The node which identifies a bandwidth power reading of 30+/- 1.05 dBm was changed to reflect a new tolerance of 25+/- 1.05 dBm.

Get change

Version: *

Mode
☒ Update and unlock
☐ Update and lock
☐ Unlock only

Item filters
☒ New
☒ Existing
☐ Unrestricted

Options
☐ Recursive
☐ Create CCC directories
☐ Prompt for description
☒ Display lock errors

OK Cancel Help

Figure 5-5. Check-in screen for changes made using CCC/Manager.

h. Reporting Capabilities

The degree to which reports are available for a project will depend upon the file types used. Reports generated on text files can identify specific lines which have been changed. Reports generated on binary files rely upon their date/time stamps as flags that a change has been made. However, there is no capability to identify what changes were specifically made.

The types of reports generated by CCC/Manager are limited to:

- Versions Report. The Versions report provides summary change information on selected items. A typical version report lists the configuration item, the configuration, and the changes by name and description for each item.
- List Change Report. This report identifies the line-by-line changes between one versions of a text file and another. This report is not available for binary files.
- List Difference report. Similar to the List Change Report, the List Difference Report flags the differences between a configuration item in a particular configuration or directory and another copy in a different configuration or directory. The List Difference Report is not available for binary files.
- Package definition report: Package definition reports describe a particular change package and the change names associated with it.
- Package history report: A package history report displays a record of actions performed on a specified package.

i. Auditing

CCC/Manager's support to the configuration auditing process is in the form of an audit log. The audit log keeps track of item updates and deletions, package creations, deletions, migrations, and promotions. Any combination of the audit options may be selected. The audit log can be important in not only the monitoring of configuration library activities, but also in the auditing of the CM process.

j. Interactive Merge

CCC/Manager has the capability to merge changes through its Interactive Merge function. The Interactive Merge function allows the development team to merge changes made in one or more versions of a configuration item with those in another. This capability can be of particular importance when concurrent development is taking place. If developers are working in parallel on separate versions of an application, several changes may need to be merged into a single version.

Interactive merges can also be used for retrofitting emergency maintenance to current development projects. For instance, if a ship were to report a problem with their version of the MK92 MAES which needed an immediate patch, one would want to

incorporate this change into the current version as well. By using interactive merging, the changes made to the patched file could be included in the current build.

k. Compatible Operating Systems

CCC/Manager and its supporting product family are compatible with the following operating systems:

- MVS/SP, XA, ESA
- VM/CMS, XA
- AIX
- VMS
- Ultrix RISC
- UNIX, SCO UNIX
- OS/2, MS Windows, and Windows NT
- Windows
- SunOS, and Solaris

This allows for the expandability and portability of a development effort and its CM program.

l. Associated Products

There are several tools made by Softool which are specifically made to enhance the features of CCC/Manager. CCC/Pro is one such tool. CCC/Pro is a problem tracking tool which is used to track a problem from its discovery, until changes to the software are made to correct it. This tool enhances the automated change control capabilities of a CM process.

Another tool is CCC/Make. CCC/Manager contains a GUI for interacting with CCC/Make. CCC/Make is a tool which will take approved changes and incorporate them into a build or release version of an application. This further integrates the CM process into the software development cycle.

m. Suitability to the MK 92 MAES Project

For basic change control functions, check-in/check-out, and version control, CCC/Manager is well suited to the needs of the MK92 Project. Additionally, its capabilities in identifying the text file changes made to the knowledge base using the script based graphical tool, allCLEAR, is also beneficial.

The interface is relatively easy to understand, particularly after one follows the tutorial in the primer. The terminology in the manual is similar to that in the CM literature, making the tool easier to understand in the context of the entire CM process.

The required change name and description upon check-in is a good way to promote change documentation. Recursive check-in enables one to set up a CM library in less than 15 minutes. The ease with which access controls are set up is another plus.

On the negative side is the tool's inability to handle the change tracking and reporting of binary files. Recognizing this is a technological problem in that one cannot dissect binary files in a manner which identifies specific changes, some features for using time stamps should be in place to identify at least a change was made.

The tool meets the requirements which are technologically feasible for the CM of the MK92 MAES. Its reporting capabilities are adequate for conducting maintenance of the knowledge base. Overall, CCC/Manager is a strong candidate for selection as the tool for the MK92 MAES project.

6. PVCS Version Manager

PVCS Version Manager (PVCS) is a configuration management product developed by Intersolv, Incorporated. The version examined in this subsection is version 5.1.1. The following is a summary of the main features of PVCS. Files from the MK92 MAES are used as examples to demonstrate the capabilities of PVCS. Descriptions of PVCS' functionality are adapted from the PVCS Administrator Guide and Reference. (Intersolv, 1993b)

a. Cost

The MK92 MAES project team was able to obtain a copy of PVCS for the educational price of \$100 per user. The regular price is \$599 per user (DeRossi, 1994).

b. Interface

PVCS for Windows is actually the command line version of PVCS which is common UNIX and DOS users with a Windows interface overlay. Through the use of ASCII configuration files, PVCS is able to tailor almost every aspect of its functionality. These files, however, use PVCS commands which can be cryptic. The purpose of the GUI is to incorporate as much of the functionality as possible from this command line interface in a graphical format using radio buttons and dialogue boxes.

There are actually two interfaces in PVCS, one for the system administrator and a second for users. The PVCS Version Manager Administrator is for the CLA to use to establish and manage the CM library. Once a project has been set up, most CM functions are launched from the Main window of the PVCS interface built for the user. Figure 5-6 is an example of the Main window for the MK92 MAES project.

Working sets are created by the CLA that make the directory paths from which files are checked out invisible to the user. Files are grouped together under a more descriptive name. In Figure 5-6, "MK92 MAES Program Files" is the working set describing those files which are part of the expert system.

In addition to the working sets described by the CLA, the user may create custom working sets. Custom working sets allow for the logical grouping of files in different archives and directories. One such use for this might be in Independent Verification and Validation (IV&V). IV&V personnel could create custom working sets which contain the Adept files for a particular module as well as the knowledge document files to which they must be traced.

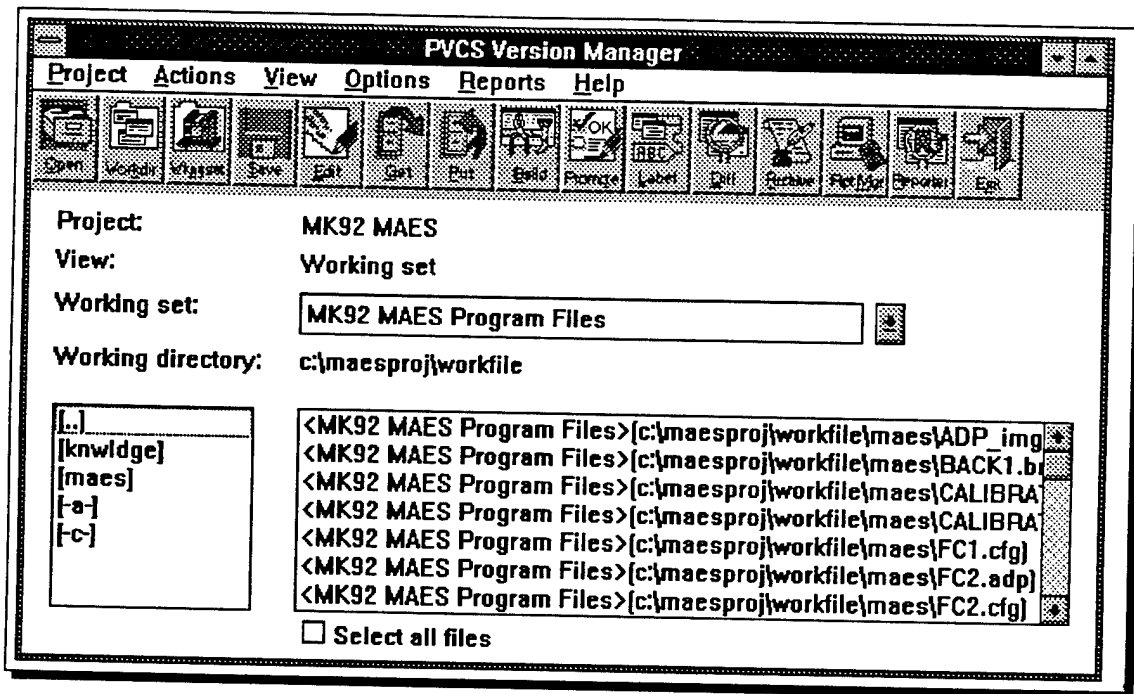


Figure 5-6. PVCS Version Control Main Window for the MK92 MAES.

c. Access Control

PVCS is capable of providing a very fine degree of access control. To accomplish this, it relies upon the use of ASCII files known as access control databases. An access control database is an encrypted file in which user groups and individual users are assigned access. Additionally, users can be assigned passwords.

PVCS provides a list of privileges which can be combined to create custom privileges. These privileges can then be assigned to entire groups or limited to certain users. Though providing for more options than CCC/Manager, project management should weigh the need for access control and the functionality provided by other CM tools against the overhead associated with administering the access control database. In the case of a small development team or project, the access controls provided by CCC/Manager and the ease with which they are implemented may outweigh the power provided by PVCS' access database scripts given the maintenance challenge PVCS' method presents.

d. Life Cycle Modeling

When the CLA creates a project, they have the option of establishing a life cycle model through which all files are migrated. PVCS refers to this as a promotion model. As with CCC/Manager, the CLA can customize the hierarchy of the promotion model to match the project's software development life cycle.

e. Check-in/Check-out

PVCS supports check-in/check-out. The user selects a base working set with which they would like to work. They may then select any combination of files from the working set to check out. Specific working directories can be chosen by the user though the creation of custom working sets, or a default working directory can be established by the CLA.

As with CCC/Manager, PVCS allows for lock out of files upon check-out. However, unlike CCC/Manager, one cannot tell by looking at the Main window which files are unlocked at a particular time. To determine what files are locked, one must print an archive report.

The delete upon check-in function clears the working directory when files are checked in. However, if a file is not modified and is unlocked rather than checked-in, PVCS does not delete the file from the working directory. This has the potential to leave files, no longer under CM, in working directories.

f. Change Control

Change control in PVCS is similar to that provided by CCC/Manager. Check-in/check-out, lock-out functions, and PVCS' equivalent to packages, labels, are virtually identical to CCC/Manager. Unfortunately, PVCS is also unable to adequately track changes to binary files. The use of custom worksets tailors the change process allowing for a grouping of files across directories, and functions.

For example, often a change to the knowledge base necessitates a change to the corresponding module in the expert system. By establishing custom worksets which

link the knowledge document of a particular module with the associated Adept modules, the change process is streamlined.

PVCS also allows one to select what text editor is executed when the Edit icon is selected from the Main window. This is particularly useful in maintaining the knowledge base. From the edit icon the tool used to document the knowledge base, allCLEAR, can be launched to make changes to the knowledge document. Switches can be set which will allow for automatic check-out of files upon launching allCLEAR and automatic check-in upon exiting the editor. This further supports the goal of creating a seamless development environment.

g. Version Control

Through the use of access controls, lock-out features, and the promotion model concept, PVCS is able to provide a development project with version control. PVCS ensures only those configuration items migrated to the product level of the MK92 MAES' promotion model are available for release.

h. Reporting Capabilities

PVCS has the capability to generate several types of reports. Additional reporting capabilities are provided when PVCS is integrated with the add-on product PVCS Reporter.

- **Difference Report.** The Difference Report displays the modifications made between two revisions or files. As with CCC/Manager, the generation of difference reports between two binary files is not very useful.
- **Archive Report.** The Archive Report is used to provide information on archives and the revisions they contain.
- **Journal Report.** The journal report is used to document the activity of an archive. The basis foundation of a journal report is the journal file. A journal file is a log maintained by PVCS of modifications made to project archives. This log records the name of the archive, the action used to modify the archive, who performed it, and when it was performed. The CLA can configure PVCS to keep a journal file.

i. Version Labels

A version label is a name that identifies a group of revisions stored in separate archives. Version labels are used to identify the revisions that comprise a version of an application. Version labeling is similar to the package function used in CCC/Manager. It allows for groups of revisions to be managed easily.

Floating version labels enable the easy identification of the most recent versions of files assigned such labels. This provides a convenient method of ensuring the latest version of an application is being manipulated.

j. Keyword Embedding

Keywords are flags in which PVCS can automatically insert data such as the version number, author, date, and time of modification to files as they are checked in. The embedding of this information is recommended by Buckley (1993) as part of the coding process. The use of an automated tool to accomplish this makes its implementation less difficult.

k. Merge

As with CCC/Manager, PVCS has the capability to merge revisions produced by concurrent development efforts. This is done through a merge dialogue box. As with the other dialogue boxes, the inability to browse for a desired file, or scroll through a long path statement to identify a file, makes the merge function cumbersome.

l. Promotion and Migration

PVCS supports promotion and migration through all phases of a promotion model from a configuration to PC directories. By selecting the Promote icon from the main menu, one can migrate files to the next highest level in the promotion model. Figure 5-7 is an example of the promotion dialogue box.

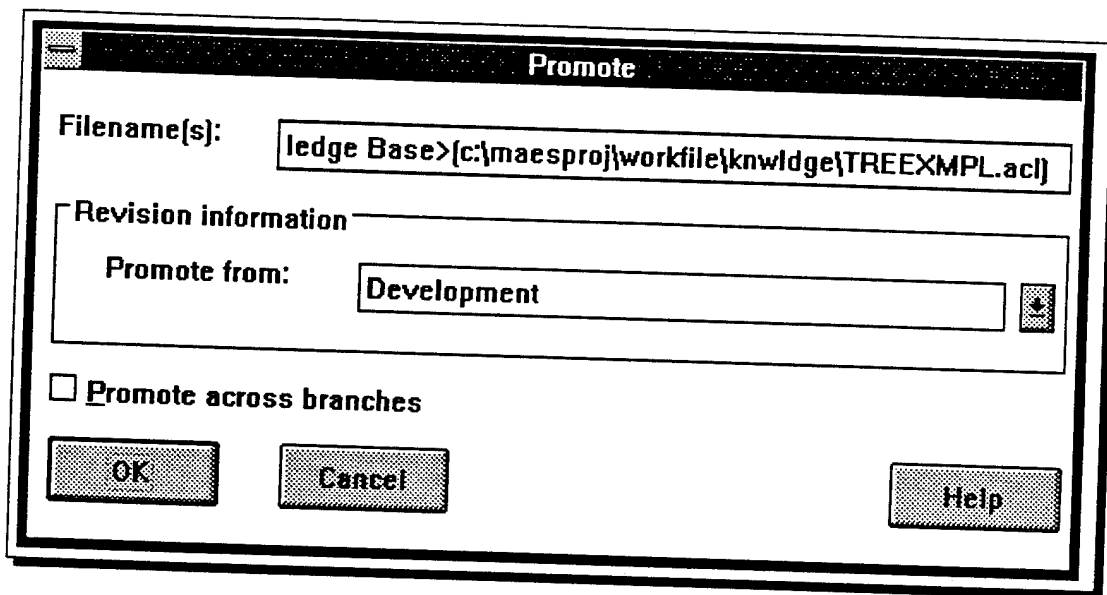


Figure 5-7. PVCS Promotion Dialogue Box.

m. Compatible Operating Systems

PVCS supports the following operating systems:

- MS DOS
- MS Windows, Windows NT
- OS/2
- UNIX

n. Associated Products

There are two products provided by Intersolv which are designed to integrate with PVCS Version Manager.

PVCS Configuration Builder is a product which interacts with PVCS to build current or previous versions from archived files.

PVCS Reporter is another add-on product which enables the project team to construct customized reports.

o. Suitability to MK 92 Project

PVCS is a powerful tool; however the user pays for this capability in terms of a steep learning curve. The use of terminology that is different from that in the literature complicates the learning process. As noted by DelRossi (1994), the GUI, though a vast improvement over the command line version of PVCS, could be better designed. It is apparent to the Windows application user, PVCS' interface was an afterthought.

Many of the program's access control capabilities come from an ASCII script referred to as the Access Control Database. Though powerful, the cryptic command lines do not promote ease of use.

Another problem with the interface is the fact the dialogue boxes remain very command line oriented. For instance, instead of providing a browse capability, the user is required to type in the entire path to files about which they desire to generate reports. The power of the Windows interface is under-utilized and thus adds unnecessary burden to the user.

Another shortcoming of PVCS is its inability to track changes to binary files. As noted in the previous subsection, this is a technological problem common to all CM automated tools. Until greater consideration is given to the integration of project management functions as integrated features of development tools such as expert system shells, this problem will persist.

Once the development team overcomes the steep learning curve, PVCS' power makes it an excellent automated CM tool. Custom working sets promise to provide an easier method for tying knowledge documentation to the associated program files. A CLA who is familiar with the access control database's nuances can introduce a high degree of control to directories and files. The degree to which PVCS can be customized, the tools which are available as add-on products, and its reputation in the marketplace further strengthens it as a choice for use in the MK92 MAES project.

While PVCS has not been ported to as many operating systems as CCC/Manager, all those likely to be used by an expert system development team for the MK92 MAES are supported.

PVCS meets the current requirements of the MK92 MAES project including the potential for distributed development. Should the MK92 MAES project evolve into the establishment of an expert system development center, PVCS will can be expected to meet any foreseeable functionality requirements.

7. Evaluate Candidate Tools Using the Chosen Evaluation Criteria

Using the selected criteria, CCC/Manager and PVCS were compared. This subsection presents a summary of the comparison of CCC/Manager and PVCS as evaluated according to the grading criteria.

Table 5.1 summarizes the comparison of CCC/Manager to Intersolv's PVCS Version Manager. In those categories where an ordinal response is provided, the following scale was used.

- Excellent. The product completely satisfies all aspects of the aspect rated in terms of functionality and ease of use.
- Very good. The product completely satisfies all aspects of either functionality or ease of use, but does not completely satisfy both.
- Satisfactory. The product meets the minimum standards in terms of both functionality and ease of use, but does not completely satisfy the needs of the project in either category.
- Poor. The product does not meet the minimum threshold in either functionality or ease of use.
- Not available. The product does not have this feature, therefore it could not be evaluated.

Without question, these selection criteria are subjective. However, given the fact the decision process is designed to select the tool for which the user has the greatest preferences, an inherently subjective undertaking, the use of subjective measures, in this instance is not seen as a limiting factor in the selection process.

Category	CCC/Manager	PVCS
Cost per user (to the MK92 MAES team)	\$495.00	\$599.00
Security	Very Good	Very Good
Change control features	Very Good	Very Good
Check-in check-out functions	Excellent	Very Good
Lock out function	Excellent	Very Good
Ability to track changes to Adept procedures and binary files	Not Available	Not Available
Version control features	Very Good	Excellent
Ease of Incorporation into the expert system life cycle	Excellent	Excellent
Ease of use	Excellent	Excellent
Interface	Very Good	Satisfactory
Ease of installation	Excellent	Satisfactory
Ease with which users can learn to use the tool	Excellent	Satisfactory
Quality of the system documentation	Very Good	Very Good
Quality of on-line help	Very Good	Satisfactory
Availability and Quality of tutorials	Very Good	Very Good
Quality of CM reporting functions	Satisfactory	Very Good
Team support	Satisfactory	Very Good
Customizability	Satisfactory	Excellent
Interoperability with other tools.	Very Good	Very Good
Release management functions	Very Good	Very Good
Build support	Not available	Not available
Performance. Time to check-out MAES file FC1.ADP (2.7 MB) from the tool's product support library in seconds. (Intel 486-DX2 50MHZ, 20MB Ram)	64 seconds	36 seconds
Overall suitability of features to the MK92 MAES CM process	Very Good	Excellent

Table 5.1 Comparison of CCC/Manager to PVCS Version Control.

8. Select a CM Tool

Based upon the analysis of the previous sections, the tool selected for incorporation into the MK 92 MAES' configuration program was PVCS Version Manager. Although the Configuration Library Administrator (CLA) encounters a steep learning curve when first configuring the product for a development effort, it is relatively easy for the programmer to operate. Features such as the capability to launch the user's favorite editor from within the tool, while at the same time providing automated check-in/check-out make PVCS particularly suitable for use in the development of the MK92 MAES.

No CM tool meets every need of a project. One significant drawback of PVCS is the need to maintain the access security database. It was determined, given the limited number of people working on the MK92 MAES project at any one time, the implementation of a user's access controls would not be an insurmountable task. It will however require more time and skill than the dialogue box used in CCC/Manager.

Another shortcoming of both tools is the technological difficulty of tracking changes to binary files. The only support available for binary files by either product centered on check-in/check-out functions, as well as version control. However, as the CM process of the MK92 MAES must also support maintenance of the knowledge base, other features needed to be considered. It is here that PVCS distinguished itself.

CCC/Manager is an extremely powerful and easy product to use. For almost any other project, its ease of use would have outweighed any perceived advantages of PVCS. Given the fact most of the MK92 MAES development process involves changes made in a visual programming environment, virtually all features useful in PVCS were available in CCC/Manager, not to mention easier to use. However, the ability to create the custom working sets is ideally suited to the challenges of applying CM to both the knowledge base and expert system software. This capability perhaps more than any other was a deciding factor in the project team's selection.

VI. CONFIGURATION MANAGEMENT IMPLEMENTATION

This chapter presents recommendations for the implementation of a configuration management plan for the MK92 MAES project. An adoption strategy is established for all four tasks of the CM. In addition, a recommendation for the implementation of a CM process for a small expert system development center is made.

A. CM IMPLEMENTATION ISSUES

One cannot blindly apply the principles of configuration management to every project in the same manner. Each project faces distinct challenges. Therefore, CM must be tailored to each project on an individual basis. To this end, this subsection addresses some issues a development team should take into consideration when developing a CM program.

The highly evolutionary nature of prototypes requires a CM plan that will be designed with flexibility as a foremost consideration. When developing an expert system, the need to apply CM is not only to the software, but the knowledge base as well. This fact multiplies the volume of changes requiring control. In addition, organizational factors, such as resistance to change or the impact of corporate culture, adds to the difficulty in implementing CM. A brief discussion of the implementation issues follows.

1. Organizational Issues

A good CM plan will attempt to achieve a fit with the organization's culture in a way that will provide the degree of control over software which is consistent with the quality goals of the organization.

In addition to the culture of the organization as a whole, one must be concerned with the effect the introduction of CM will have on individual project team members. Some team members may feel as though CM will restrict their creativity. Others may resist changes to their organization. These barriers must be overcome if a successful CM process is to be initiated.

2. Skill level of Organizational Personnel

The skill level of an organization's personnel is another important consideration when implementing a CM plan. The degree to which developers are familiar with the concepts of CM must be a consideration. Personnel familiar with CM and its potential benefits are more likely to adopt a CM plan than those unfamiliar with the concepts. Training in CM may be necessary to overcome any deficiencies.

3. Organization's Level of Process Maturity

Process maturity refers to the processes and software design policies of the organization. An organization that has already implemented control policies is likely to be more receptive to the introduction of configuration management. CM would simply be an enhancement to existing policies and procedures. However, if the organization does not have any policies, or has inadequate or informal policies, the task becomes more daunting. A formalized CM program will need to be established. Education for the developers and management at a fundamental level will be necessary.

4. Technological Issues

As was demonstrated in Chapter V, today's software environment precludes the development of a generic, off the shelf tool that will fit every user's needs. There is no "silver bullet" which will provide the perfect automated solution to configuration management. What is available is a myriad of tools for different platforms and different project sizes; each providing markedly different levels of functionality.

A CM program must match not only the technology that is to be applied to a project, but also be compatible with the CM issues presented by the software under development. For instance, if one were to implement a CM plan on an expert system one should take into account the change control of the expert system software, as well as the changes to the knowledge.

5. Resources Available

The availability of resources is another concern. The implementation of a CM plan may require training, additional personnel, automated CM tools, and other resources. The degree to which a CM plan can be implemented may be dependent upon the availability of time, money, and other constraining resources. The CM plan should be designed so that it is sustainable given the resources likely to be available to support it.

B. PHASES OF CM ADOPTION

Before presenting the implementation recommendations for the MK92 MAES project, it is useful to describe a generic process for CM program implementation. As recommended by Dart (1993), one should treat these phases as though they are starting points of a CM checklist. Phases one to three need not necessarily be conducted in series. Depending upon manpower availability and the complexity of the project, the steps presented in the following paragraphs could be undertaken simultaneously.

1. Phase 1: Determine CM Status and Needs

In the first phase, the organization must determine the extent to which CM is being practiced within the organization. To do this, it may first be necessary to educate the project members on CM. It may be that some degree of CM is already being practiced, but not in a well defined manner. A review of the organization's procedures and current practices is useful in determining the degree to which CM is being practiced.

Once the developers know what the status of their CM process is, they should identify what their CM needs are. Complex, mission critical software projects using distributed development techniques will require a more rigorous CM processes than a small development team building work productivity tools on a single computer.

2. Phase 2: Evaluate Candidate CM Tools

As the project team identifies its needs it can begin identifying and evaluating automated CM tools which will meet their needs. The decision as to whether or not the project development team requires an automated tool should be made. A small one or

two person coding effort may not necessitate the functionality of a CM tool. Chapter V discussed the considerations for evaluating CM tools in greater detail.

3. Phase 3: Write the Configuration Management Plan

The configuration management plan serves as the road map for the configuration management process. The plan adopted should match as closely as possible the needs, culture, and capabilities of the project team. A plan that is too restrictive, does not satisfy critical needs, or is beyond the capabilities of the development effort will not be adopted by project personnel.

4. Phase 4: Implement a Pilot Project

If practical, the CM plan should be initiated on a pilot project. By doing so, the team is able to elicit feedback from developers and make adjustments to CM processes before the CM plan is applied project-wide. This may ease the transition to the new CM methodology.

5. Phase 5: Implement CM Plan

Once the pilot project has been implemented and adjustments to the CM plan have been made, the CM process should be extended to the intended project. Every effort should be made to make this implementation as smooth as possible. Training for developers, management support of the effort, and the availability of resources are all critical factors for a successful implementation (Buckley, 1993).

6. Phase 6: Evaluate and Adjust Plan as Necessary

Just as software evolves, so should the project's CM plan. Through the use of feedback from developers, management, CM library administrators, and customers, the configuration manager is able to recommend changes to the CM plan which will streamline the process and ensure the success of the CM plan.

C. CM ADOPTION PHASES OF THE MK92 MAES

This section addresses the adoption phases of a CM plan for the MK92 MAES project.

1. Phase 1: The Status of CM and Needs of the MK92 MAES Project

a. Status of CM in the MK92 MAES Project

The first step in the implementation of the MK92 MAES project's CM plan is to identify the current status of CM in the project as well as its CM needs. The MK92 MAES project is presently at what Capers Jones (1994) describes as severity level one. That is, neither automated nor manual configuration control have been formally established. Changes are made to the expert system software as they are received by NSWC-PHD or when errors are found. Only when a change requires a significant revision to the project is there any discussion among faculty and students as to the feasibility and impact of proposed changes.

No formalized process for tracking changes is currently in place. In several instances this has resulted in changes to knowledge documents being stored by a developer without the knowledge of other project members. This situation has caused problems, particularly with the continual student turnover.

Version control for the MK92 MAES consists of unprotected directories in which the latest version resides. There are no safeguards to prevent someone from making unauthorized changes to release versions. Archival of versions consists of storing knowledge documentation in loose-leaf binders which contain a copy of that revision's knowledge documentation, diskettes containing the program's code, and a paper copy of the implemented procedures. Though far better than no configuration management at all, there is no process by which decisions are made to create new releases and archive older versions.

Having noted the CM shortcomings, several factors should be pointed out. Expert System CM is in its infancy. The vast majority of expert system projects have not

implemented a CM plan. As the literature review demonstrated, there is a scarcity of writing on the subject. Also the MK92 MAES project was originally conceived as a prototyping demonstration, for which the establishment of a CM process was considered too costly. As Chapter IV noted is often the case with expert system development, it was not until well into the project that a change in strategy resulted in the evolution of the demonstration prototype into a developmental effort aimed at delivering a production fieldable product. This study began a year ago when the aforementioned change in strategy was established. In that time, three groups of students have transferred. The resulting loss of corporate knowledge has underscored the necessity of implementing a CM process.

b. CM Needs of the MK92 MAES Project

The CM process developed for the MK92 MAES must be flexible enough to process changes from many different sources in a timely manner. This flexibility is crucial, particularly during the developmental stages of the expert system life cycle. Proposed changes must be evaluated and acted upon with as little delay as possible.

The CM process should account for the work environments of both NPS and NSWC-PHD. The development atmosphere of an academic setting and the fact most implementation is done by Master's Thesis students must be taken into consideration. For instance, the formalized, highly structured, boards typical of organizations with strict hierarchies such as NASA are inappropriate for the academic setting of NPS. By fitting the process to the culture and procedures of the two organizations, disruptions to the developer are minimized during initial implementation of the CM process.

Strict access control, version control, and check-in/check-out procedures are required for the MK92 MAES project. Many of the CM-related problems encountered (i.e., identifying latest versions of files) during the development of the first two modules of the MK92 MAES could have been avoided through these establishment of a CM process which encompassed these features. The MK92 MAES project needs a CM process which can control not only the expert system, but also the knowledge base. The

ability to archive both in an automated fashion and provide change information and functionality is highly desirable.

Current technology does not support automated change control for the expert system development shell, Adept, used in building the MK92 MAES beyond the use of date/time stamps. Some method which is capable of tracking the status of proposed changes should be adopted. Specific changes to nodes may need to be identified using comments or through the use of a manual process. Whichever process is implemented, it needs to be as easy as possible to use. The frequent turnover of students necessitates a process with an easy a learning curve. It must also be affordable. The benefits gained from the implementation of a CM program must outweigh the associated overhead. The need to minimize overhead is particularly important in the early phases of CM implementation when the greatest resistance is likely to be met.

2. Phase 2: Evaluate Candidate CM Tools for the MK92 MAES Project

Chapter V discusses the evaluation and selection process of an automated CM tool. The tool selected, PVCS by Intersolv, encompasses a degree of functionality that promises to address many of the requirements of the MK92 MAES' CM plan. Examples used in this chapter are in terms of the process as implemented by PVCS. Adjustments will need to be made to the plan to account for the use of an automated CM tool other than PVCS Version Manager.

3. Phase 3: Write the CM Plan for the MK92 MAES Development

The remainder of this chapter is dedicated to presenting specific recommendations for application of CM to the MK92 MAES development. Subsection D, that follows, addresses recommendations for a CM plan for the MK92 MAES. Specific focus is on the four tasks of configuration management and their applicability in managing changes to the MK92 MAES.

4. Phase 4: Implement a Pilot Project

As there was only one expert system project under development by the MAES development team, it was not possible to initiate a pilot project. However, the various tasks of the CM plan, such as change control and status accounting, were practiced with PVCS using MK92 MAES software files and knowledge documentation implemented in allCLEAR. This included the following:

- establishment of a configuration management library using PVCS functions (such as working sets)
- CM library administration
- creation of expert system life cycle stages using PVCS promotion models
- populating the CM library
- editing files using the tool's check-in, check-out functions
- creating reports for review
- reviewing change information
- using the tool's journal
- migrating packages through the promotion models
- promoting of tested modules to the status of a product.

By using this approach the project development team was able to devise a CM implementation that maximizes the capabilities of the selected CM tool.

5. Phase 5: Implement the MK92 MAES CM Plan

At this point, it is important to note that no one aspect of configuration management can prevent poor software development practices. Configuration management is, in itself, a system. Only through a combination of CM tools, policies, procedures, audits, and personnel discipline can an organization expect to adopt an effective CM program. Plan implementation begins with the establishment of CM policies and the education of project team members. Unless they understand what the potential benefits of CM are to them and their role in the CM process, one cannot expect a CM

implementation to be successful. This thesis should be an integral part of the CM training process for future MK92 MAES project team members.

To implement the MK92 MAES CM plan, the assigned configuration library administrator (CLA) needs to establish the CM library and working libraries for the developers. Access controls were implemented, and archives populated with the baselined versions of software under development. PVCS automates many of the CM functions. The use of PVCS in the implementation process will be identified in greater detail through the use of examples provided later in this subsection. Once the appropriate controls were in place, and the CM library established, the programmers can begin developing software and managing domain knowledge using the new CM process.

6. Phase 6: Evaluate and Adjust Plan as Necessary

No implementation will be flawless. Problems which were not anticipated by the drafters of the MK92 MAES CM plan are bound to arise. Additionally, as the MK92 MAES project evolves, so to will the CM plan. Since the CM plan for the MK92 MAES has only been in place for a short period of time, insufficient feedback has been generated to evaluate its suitability at this point. However, it will be necessary for future MK92 MAES project team members to provide feedback to the configuration manager, CLA, and project manager as to their perceptions of the CM processes strengths and weaknesses. Their recommendations for improvement will be critical in ensuring a CM process that promotes software integrity is adopted by the project team.

D. CM FOR THE MK 92 MAES

Typically, a configuration management plan is a document which includes an overview of the purpose, tasks and roles of the CM process. Due to the desire to limit redundancy, yet provide the project team members with a working document, the background information which would normally be presented in a CM plan should be considered to have been presented in the preceding chapters. This subsection outlines the policy recommendations for CM in the MK 92 MAES project. Terms such as "will" and "shall" are in keeping with the traditional format of CM plans and represent policies in

which an action is required. Terms such as "could", "should," and "may" are used in situations where the policies presented are recommendations to project management.

1. CM Organization

Although some responsibilities and duties are addressed here, many duties and responsibilities are outlined in the recommendations regarding the specific CM tasks.

a. Project Manager

The project manager (PM) for the MK92 MAES should be a faculty member. This faculty member will act as the approval authority for actions such as the granting of user accesses, change approval, and change migration. The PM should promote the improvement of the CM process as well as ensure developers are adhering to established policy. Additionally the PM will be responsible for allocating resources to the developers.

It may be necessary to have an additional faculty member act as the program manager in charge of project development. If such a position is created, approval actions listed in the following sub-sections should be the responsibility of the PM in charge of development.

b. Student Project Leader

The MK92 MAES project team should establish the position of student project leader. This will allow the faculty acting as project manager to delegate some responsibilities regarding reporting and approval requirements. The degree to which responsibilities are delegated should be considered with caution. The faculty member is ultimately responsible for the project. A clear definition of responsibilities regarding issues which could adversely affect the project should be made. The project manager may wish to retain control over decisions to baseline, change control authority, and version control. However, approval authority for granting access controls could be delegated. The decision to delegate responsibilities should rest with the project manager and be based

upon the number of students in the project and the need for supervision of the development effort.

c. Configuration Manager

The configuration manager for the MK92 MAES should be either the student project leader, the faculty member in charge of development, or another responsible developer. The current size of the project does not warrant the establishment of a separate position, therefore it should be assigned as collateral duty. The configuration manager must be the authority on MK92 MAES CM policy. In addition to the various responsibilities outlined in this chapter, the CM manager should be evaluating the emerging trends in CM and their applicability to the evolving needs of the implementation.

d. Configuration Library Administrator

The CLA is responsible for creating the various CM libraries outlined in Chapter V. Additionally, the CLA will establish and maintain the PVCS archives, working sets, and access controls. It is recommended the CLA not be given the authority to approve the addition or deletion of archives; only the responsibility and access for implementing the decisions of the project manager and configuration manager regarding these matters. This ensures project management is aware of major changes to the CM library.

It is recommended that the position of CLA be assigned to a student developer. If student's are involved in the IV&V of the expert system, their role in the product assurance process would make one of them the logical choice for managing the configuration management library.

As pointed out in Chapter V, PVCS has two interfaces. The CLA shall perform library administrative functions through PVCS version control administrator interface. All programming functions should be used through the user interface. This will prevent users from accessing CLA functions.

e. Configuration Control Board

As pointed out in Chapter V the configuration control board (CCB) acts as the clearinghouse for all change requests. For the MK92 MAES project, it is recommended the CCB is made up of a faculty member, the student developer responsible for implementing the proposed changes, and any project personnel in a position to evaluate the impact of the proposed changes.

The CCB for the MK92 will meet as an integral part of the development team's scheduled project meetings, preferably before the discussion of other business. Other meetings of the CCB can be called as needed. This approach is best suited to the project environment of the MK92 MAES. It allows the students and faculty to meet weekly or bi-weekly only, thereby reducing the burden of project meetings upon their other responsibilities.

The CCB's focus is on the evaluation of a proposed change's impact and how to allocate resources, if necessary, for its implementation. It is important the CCB remain focused on this process and not drift into discussions of other project related matters. It is the responsibility of the project manager (PM) to ensure the CCB meeting deals exclusively with change related issues.

f. Programmers

The term programmer refers to any member of the project team involved in the physical coding process of domain knowledge into Adept procedures. Project management and the CLA should not be given access controls to make changes to specific nodes. CM requires discipline in development, and that discipline must begin with the identification of who is allowed to make changes.

A primary responsibility of a programmer is to be familiar with and follow the established procedures of CM for the MK92 MAES. Through the use of access controls, access can be limited to specific files, archives, and functions. It is the responsibility of programmers to use only the access for authorized programmers they have been assigned. For instance, a programmer may be assigned the additional

responsibilities of configuration library administrator (CLA). It is the responsibility of the programmer to avoid using their CLA access to make modifications to code.

One solution may be to remove programming functions from the CLA's access level. When performing CLA functions, a programmer will have to use a different account than when they are making changes. The power of the CLA's access and the ability to not only make changes to files, but delete entire archives, is the reason for making the decision to segregate responsibilities and access. The inadvertent deletion of an archive because the programmer thought they were deleting their own work-in-progress must be guarded against.

g. NSW-CPHD and Domain Expert

NSWC-PHD is presently the collection point for changes made to the MK92 MOD 2 FCS. This role makes them the logical collection point for trouble reports from the fleet regarding the MK92 MAES. Therefore, NSW-CPHD should ensure the expert system's domain knowledge is updated. Until the use of the script-based graphical tool, allCLEAR, to document knowledge is completely implemented, changes to domain knowledge should continue to be provided to the MK92 MAES development team at NPS in the form of hand-drawn diagnostic trees. These trees should then be incorporated into the allCLEAR coding process of the domain knowledge. Once allCLEAR implementation has been completed, NSW-CPHD should make changes to the domain knowledge using allCLEAR. This will speed up the change implementation process.

The ability for NSW-CPHD engineers to make corrections and changes to domain knowledge makes the reporting of problems as to fleet readiness and safety more efficient and effective. An assessment of the impact of a problem with the MK92 MAES should be included with any changes to the domain knowledge NSW-CPHD provides to the development team.

h. Customers

For purposes of this discussion a customer is defined as a ship or installation to which the MK92 MAES has been deployed. The only responsibility

assigned to the customer is to provide feedback regarding any problems associated with the MK92 MAES or recommendations for improvement. It may be necessary to establish, through NSWC-PHD, a reporting procedure regarding the use of the MK92 MAES and any recommendations or problems encountered. Customer feedback is crucial if improvements to the MK92 MAES software is to be improved.

2. CM Library

The CM library for the MK92 MAES will be established and maintained by the configuration library administrator. The current method of implementation of the CM library will be to use the automated tool, PVCS, described in detail in Chapter V. All references to specific implementation of the CM library will be in reference to PVCS and its functionality. The reader is referred to Chapter V, or the PVCS Reference Guide (Intersolv, 1993a) for a description of the specific functions and terminology used in this discussion of the CM library.

Two base working sets are recommended for the MK92 MAES. One base working set should be established for the MK92 MAES project software. A second base working set should be implemented that will contain the knowledge document as it is represented using the script-based graphical tool allCLEAR.

In addition to the working sets for the domain knowledge and product software, it is recommended that CM working sets be established for documentation (if digitized) and the Installit build-script files. The adoption of automated tools such as PVCS Make (a build program by Intersolv that integrates with PVCS Version Manager) and document management software may be one option for simplifying this process.

The CLA can establish whether or not a working set will have public access or not. If the project manager wishes to restrict programmer's access to certain base working sets, such as the Installit build-scripts, he can tell the CLA to create the base working set, but not select "public" on the working set creation screen of PVCS.

Within each working set, archives shall be established by the CLA for every file checked-in . This is done as part of the file check-in process of PVCS. The archives will be used to store previous versions of files.

In addition to the base working sets, the CLA can also establish custom working sets which can be accessed by all programmers. This is useful for those conducting IV&V. The CLA can establish custom working sets for IV&V personnel that match files containing domain knowledge with the corresponding MAES program file. By so doing, the IV&V personnel can easily check out files which are from separate base working sets in a way that is well suited to their process. Additionally, programmers can make changes to domain knowledge and Adept procedures more easily by being able to check out the applicable files together. The establishment of publicly accessible custom working sets that correlate domain knowledge files coded in allCLEAR with program files from Adept is highly recommended.

In order to implement the life cycle of the expert system's development process within PVCS, it will be necessary to establish a promotion model. A simple promotion model that promotes files from a developmental configuration to a test library and in turn to a product library should be adequate for the purposes of the MK92 MAES. All programming and change implementation should take place in the developmental library. All IV&V functions should be performed using the test library. All builds and product disks should be created using the product library. The CLA is also responsible for establishing the promotion model to be used by the MK92 MAES project.

3. Configuration Identification for the MK 92 MAES Project

This subsection contains recommendations for configuration identification of MK92 MAES configuration items.

a. Baselines for the MK 92 MAES

Baselines for the MK92 MAES are broken down into two primary categories. The first category is the knowledge baselines. These are discussed in detail in Chapter IV, and therefore will only be listed here.

- **Functional Knowledge Baseline.** The functional knowledge baseline shall consist of the knowledge documentation representative of the MK92 MAES version 1.0 knowledge base. As additional releases are made, the knowledge which makes up the product's knowledge base shall make up the functional knowledge baseline of the subsequent development efforts. For example, when version 2.0 is released, its knowledge documentation will be used as the baseline for development efforts after version 2.0's release.
- **Developmental Knowledge Baseline.** The developmental knowledge base shall consist of the functional knowledge baseline of the MK92 MAES version 1.0 plus any implemented changes to the knowledge base after the release of version 1.0. As outlined in Chapter IV, the developmental knowledge baseline represents the current status of implemented, approved domain knowledge. Changes to the knowledge base are made to the developmental knowledge baseline.
- **Product Knowledge Baseline.** The product baseline shall be established as the knowledge documentation representative of the MK92 MAES version 1.0. (the version currently being released to test sites). Subsequent product knowledge baselines shall incorporate the knowledge representative of the latest release of the MK92 MAES.

The second category of MK92 MAES baselines is the software baselines. The reason for segregating the knowledge and software baselines was explained in Chapter IV. Software baselines are examined in Chapter III. The following is a listing of the software baselines established for the MK92 MAES.

- **Functional Baseline.** The MK92 MAES version 1.0 functional baseline shall consist of the Engineering Development Model (NSWES, 1992) and the functional knowledge baseline for the MK92 MAES version 1.0. Subsequent versions should include that documentation that defines the required interfaces and the functional knowledge defining the current release.
- **Developmental Baseline.** The developmental baseline for (current software, and current knowledge documentation) the MK92 MAES version 1.0 represents the current status of development. Therefore, it should include the latest copy of all files, documentation, and knowledge (in the form of the developmental knowledge baseline) that represents the current implementation of the MK92 MAES. As the software evolves, so to will the developmental baseline.
- **Product Baseline.** The MK92 MAES product baseline consists of the software which makes up the Adept code for version 1.0, the build scripts for making product diskettes, the MK92 MAES product knowledge baseline, user's manuals, and any other documentation that defines, describes, or facilitates the

reconstruction of version 1.0. Subsequent versions must be baselined as a result of agreement between MK92 MAES project management, and NSWC-PHD.

The version numbering of configuration items within baselines that are stored on a computer shall be accomplished using the numbering features of PVCS. Numbering of other configuration items will be discussed as appropriate in the remaining subsections. Any changes to the numbering scheme adopted here must be approved by the CCB and the faculty project managers.

b. Categorization of MK92 MAES Software

A description of the software categories for the MK92 MAES is presented in Chapter III. The following is a listing of the categories and software under each.

Category 1: Product software. This encompasses all files that can potentially be incorporated in a release version of the MK 92 MAES.

Category 2: Vendor-provided product development software. Software that falls under this category includes:

- Adept version 2.2
- allCLEAR version 2.0a
- Installit for Windows 4.59w
- PVCS Version Manager 5.1.1
- PVCS GUI for Windows 1.0

Category 3: Vendor provided software not specifically related to product development (e.g., operating systems and word processors.). Software that falls under this category includes:

- MS DOS 6.2
- Microsoft Windows 3.1
- Ami Pro 3.0

Category 4: Test software. This consists of any formal test routines written by the development team. Currently, there is no software that falls under this category.

Category 5: Other software. As the name implies, this is a catch-all category for software not falling into any of the above categories.

c. Configuration Identification Naming Conventions by Classification

Level of Software

Category 1: Product software. Since version 1.0 was established prior to the implementation of CM and build scripts have already been constructed, the names already given to MK92 MAES files should not be changed until a second release is being considered. Upon product baselining of the next release, the MK92 MAES project team may wish to migrate the naming of files to be more in line with the generally accepted formats outlined in Chapter III and presented by Buckley (1993). The program's name shall be "MK92 MAES v x.x" where "x.x" is the version number. The following is an example of the recommended naming convention for Adept files.

MAES_MODULE.FILE EXTENSION; VERSION NUMBER

MAES is the program name and should be present in all MAES files, if practical. The MODULE name refers to the module that is being coded (i.e., FC1, FC2, FC4, CAL, etc.). The label for the FILE EXTENSION will be assigned by the development tool (Adept, or All Clear) and the VERSION NUMBER will be assigned by PVCS.

Category 1 program and file names shall be recommended by programmers and approved by the faculty member in charge of development.

Category 2: Vendor-provided product development software will use the name assigned by the vendor.

Category 3: Vendor provided software not specifically related to product development. As with Category 2 and Category 3 software will be assigned the name used by the vendor.

Category 4: Test software. Test routines as a rule are not applicable to the Adept development environment. However, given the potential for a MK92 MAES programmer to develop test routines, it is necessary to recommend an identification scheme. Informal test software developed as one-time-only test routines should be named an appropriate name by the developer and should remain in the programmer's working library.

Those routines that are to become part of a formalized testing process should be labeled in the following format:

TEST_MODULE.EXTENSION; VERSION NUMBER

TEST identifies the file as being a test routine. The programmer should insert the name of the module for which the test routing was developed where MODULE appears in the example. As with the naming of product files, the EXTENSION and VERSION NUMBER are named by the computer software.

Category 5: Miscellaneous software. Any software not falling under the previous categories is not brought under CM for the purposes of the MK92 MAES project. The naming of Category 5 software is done at the discretion of its developer.

d. Configuration Identification Naming Conventions of Knowledge

The recommendation for naming the knowledge document assumes the diagnostic trees for the MK92 MAES will be implemented using allCLEAR and will mirror the implementation of the knowledge in Adept. The process used for naming software files for Adept would apply equally to the naming of allCLEAR files. The only difference would be the name of the file extension which is assigned by the computer.

e. Configuration Identification of Documentation Other than the

Knowledge Document

The use of a document identifier number such as those presented in Buckley (1993) or Chapter III is not recommended to satisfy the current needs of the MK92 MAES project team. For technical manuals, PMS cards, ORDALTs and other documentation, the MK92 MAES project team shall adopt the name given by the originator of the document. Documentation should be identified by the name given by the originator and the date drafted. If no date is on the document, the date the document was brought under CM should be assigned.

Documents originated by the MAES project team shall have their titles approved by the project manager. This is done during the normal editing process of the document.

Version description documents are to be labeled using the following format. Program name and version number are self-explanatory. DATE refers to the date the document was approved by the program manager. For example:

PROGRAM NAME-VERSION NUMBER-DATE
MK92 MAES-v1.0-3FEB95

f. Configuration Identification of Erasable Electronic Media

Labeling conventions of electronic erasable media such as floppy diskettes and magnetic backup tapes shall be in accordance with the method presented in Chapter III. An electronic media label shall contain the following information.

- Name of the contents
- Version number
- Date prepared
- Name, telephone number, mailstop, and organization of preparer of the item

- Number of the version description document (VDD) that specifies the media's contents

The following is an example of an electronic media label for the MK92 MAES.

MK92 MAES Version 1.0 04 March 1995 P.G. Metzler (408) 656-3626 MK92 Project Team Naval Postgraduate School Monterey, CA 93940 VDD: MK92 MAES-v1.0-001
--

Figure 6-1. Example of an electronic media label for the MK92 MAES.

4. Configuration Change Control for the MK 92 MAES Project

This subsection addresses the procedures and policies for configuration change control in the MK92 MAES project.

a. Change Control Authority (CCA)

The CCA for the MK92 MAES is organized by baseline. Table 6.1 is a summary of the MK92 MAES baselines and associated change approval authority.

Baseline	Approval Authority
Functional Baseline (knowledge and software)	Agreement between NSWC-PHD and MK92 MAES project management
Developmental Baselines (knowledge and software)	MK92 MAES Project Manager as recommended by CCB
Product Baseline (knowledge and software)	Agreement between NSWC-PHD and MK92 MAES project management

Table 6.1 Change Control Authority for the MK92 MAES.

b. Change Control Process

Feedback from the fleet regarding problems and recommendations for enhancement should be collected by NSWC-PHD and recorded on a change request. Change requests should be provided to NPS with an estimation of the problems impact should it not be implemented. Change requests should also accompany changes to knowledge resulting from ORDALTs and other technological changes.

Problems discovered by developers are documented on a change request form with an impact assessment and recommendation for corrective action if known. The change request is presented to the CCB at a project team meeting. The CCB will determine the impact of the proposed change and make a recommendation as to how to proceed. Final change authority will lay with NPS faculty members in instances which do not alter functionality or cause the product to vary from specifications. If the project's schedule is severely impacted, or the product's functionality is altered to the extent it will vary from the design agreed upon, then NSWC-PHD should be consulted for concurrence.

Approved change requests are to be prioritized by placing a due date for completion. The use of nominal ratings (high, medium, low priority) should be avoided. Changes will be made by checking-out the appropriate knowledge document or software module from the applicable PVCS archive. It is recommended the CLA establish working sets that group MK92 MAES modules implemented in Adept with their associated knowledge implemented in allCLEAR. This will ensure changes are implemented to both the knowledge base and the Adept procedure. Upon completion of the implementation, the configuration item should be checked-in to the archive from which it was removed. An example of the change process to both the knowledge base and software is provided in Subsection E.

c. MK 92 MAES Change Control Documents

MK92 MAES change requests and the action taken on them should be recorded on one concise document. This minimizes the amount of paperwork. Appendix

B provides an example of a change request document that could be used for the MK92 MAES.

Changes documents should be maintained in a Change Log Book that is comprised of three sections

- Proposed changes. This section contains all change requests that have not been reviewed and approved by the CCB.
- Active changes. This section of the Change Log Book contains all change requests that have been approved for implementation but have not been completed.
- Closed changes. This section contains those change requests that have been implemented or were not approved when reviewed by the CCB.

It is recommended that a problem tracking software or documentation management tool be identified and purchased to automate the tracking of change documentation. The use of an automated tool promotes the implementation of a seamless CM process. Given the size of the MK92 MAES development team, there is a need to reduce the administrative burden of the change control process.

d. Access Controls/Privileges

PVCS refers to access controls as privileges. Privileges for the MK92 MAES are approved by the MK92 MAES project manager or student project leader, and are implemented by the CLA. Table 6.2 lists the recommended privileges for the three primary user groups: programmer, project management and configuration library administrator. An explanation of each privilege available in PVCS can be found in Intersolv manual (1993a). As stated in Chapter V, custom access controls/privileges can be implemented by the CLA. The creation of custom privileges and the addition of new user groups, such as "IV&V" or "Test and Evaluation", should be approved by the project manager and implemented by the CLA.

The project manager may desire to restrict the privileges of the CLA to exclude those functions accessible to programmers. This would be done to prevent a programmer who is also the CLA from using his/her CLA account to circumvent

established controls. Table 6.2 provides recommendations for CLAs that are also programmers.

User Group	PVCS Access Controls
Programmer	AddVersion, ChangeWorkfileName, Get, Lock, ModifyDescription, Put, ModifyVersion, ViewArchive
Project Management	Unlimited, NoDeleteRev, NoBreakLock, NoChangeCommentDelimiter, NoChangeOwner, NoChangeProtection, NoInitArchive
Configuration Library Administrator (non-programmer)	SuperUser
Configuration Library Administrator (CLA is also a programmer)	Unlimited, NoGet, NoPut

Table 6.2 Recommended Access Controls by User Group.

In addition to privileges, the CLA may also establish login names and password protection. These are highly recommended to prevent unauthorized access of the MK92 MAES archives. Passwords should be created by the individual user and implemented by the CLA.

5. CM of the MK92 MAES Knowledge Base

This subsection makes recommendations for the application of CM to the MK92 MAES knowledge base.

a. CM Issues Related to MK92 MAES Knowledge Acquisition

Chapter II introduced the knowledge acquisition methodology used by the MK92 MAES project team. The traditional knowledge acquisition process, is one in which a domain expert's knowledge is elicited through interviews, documented, then tested in an iterative manner. The MK92 MAES project's diagnostic tree method of knowledge documentation, on the other hand, does not put a knowledge engineer directly into the knowledge acquisition process.

Just as the programmer has a working directory for code "in progress," the domain expert, through the absence of a knowledge engineer, is given the same luxury. Where a domain expert's knowledge acquired using traditional knowledge acquisition techniques would be brought under CM when documented, regardless of its relevance, the rate of change is decreased through the use of formalisms such as diagnostic trees. Using the knowledge acquisition technique of the MK92 MAES project, the domain expert is able to manipulate his/her knowledge at will, until satisfied with its content. The diagnostic trees are not placed under configuration management until turned over to the development team for coding.

This approach to knowledge acquisition represents a change management challenge. The configuration management process must include the domain expert. Because domain experts may not be well versed in software engineering disciplines, it is important to educate them as to the purpose and importance of configuration management. Policies and procedures must take into account the need to apply CM to the domain expert's role in the expert system development process.

b. The MK 92 MAES Approach to Implementing CM on the Knowledge Base

The use of the diagnostic trees to document the domain experts knowledge simplified the task of applying configuration management to the knowledge base of the MK92 MAES. Originally, the domain expert would draft each diagnostic tree by hand. Each diagnostic tree diagram would be connected to other diagrams using connector symbols. NSWC-PHD engineers would make changes to knowledge by hand, either by lining out errors or redrawing the affected branch of the diagnostic tree.

As the knowledge base grew and time progressed, changes became more difficult to implement. Changes were time consuming and not easily archived. Older versions had to be stored on paper in filing cabinets. Changes to knowledge documents were getting lost and, on several occasions, were not implemented. Additionally, errors

would be introduced due to misinterpretations of a domain expert's handwriting during the coding process.

To alleviate these problems, the decision was made to use an automated script-based flowcharting tool to capture and represent the domain expert's knowledge in the form of diagnostic trees. Originally intended as a tool to save time and provide a less labor intensive method of developing the diagnostic trees, the software proved to be a valuable tool in the establishing a CM process for the knowledge base.

c. AllCLEAR

The tool selected for use in the MK92 MAES project was allCLEAR version 2.0a, by CLEAR Software. AllCLEAR is a script-based flow charting program which automatically draws different types of charts by using a simple script language. Figure 6-2 is an example of an allCLEAR script. Figure 6-3 is an example of the flowchart generated by the script in Figure 6-2.

From A1.2.

Is The RF Power Output of Mixer Within -13.6+/- 1dBm at 30MHz +/- .1MHz?

(Yes) Is the RF power input to the Stalo Loop Assembly within -13.6+/- 1dBm at 30 MHz +/- .1MHz?

(Yes) Replace stalo Loop Assembly UD412/A1A8

(No) Replace/Repair Cable Assembly P1W33P2

?end

(No) Is the RF Power Output of the Stalo Assembly within +10 to +13 dBm at 8.93 to 9.43 GHz?

(Yes) Is the Stalo RF Power input to the Mixer within +9.8 to +13dBm at 8.93 to 9.43 Ghz?

(Yes) Replace Down Converter Mixer UD412/A1A3-U1

(No) Replace/Repair Cable Assembly P1W31P2

?end

(No) Replace Stalo Assembly UD412/A1A3-A3

?end

Figure 6-2. Example of a script written in allCLEAR.

The ease with which one can build flowcharts is a vast improvement over the original manual process used by the MK92 MAES domain experts. Additionally, the scripts allow for changes to be made quickly and easily.

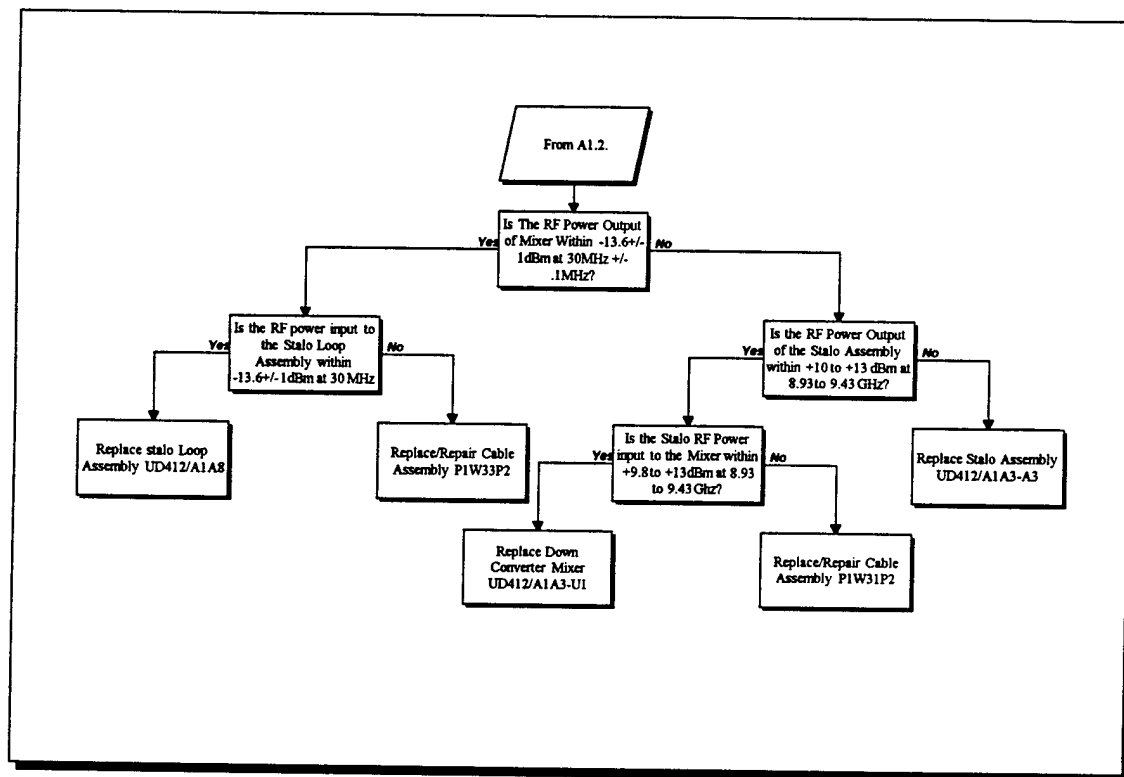


Figure 6-3. Graphical representation of a diagnostic tree using allCLEAR.

An unexpected advantage of using allCLEAR to document the expert's knowledge is the ease with which configuration management may be applied to the knowledge base. Because the diagnostic trees are implemented using a text-based script language, the features of configuration management tools which implement version control, change control, status accounting, and configuration auditing can be used to apply CM to the knowledge base. Differences between versions can be identified down to the specific changes made to the knowledge. Older versions can be recreated, and change histories can be maintained. The features, capabilities, and limitations of CM tools are discussed in Chapter V.

Because of the selection of diagnostic trees as the knowledge acquisition method, knowledge acquisition and representation phases are accomplished simultaneously. Diagnostic trees are placed under configuration management when they have been delivered to NPS for implementation. Configuration management at the domain expert's level is limited to the use of naming conventions and basic access controls to prevent work-in-progress from being inadvertently included in completed diagnostic trees.

In instances where hand-drawn diagnostic trees are delivered, a copy is made, dated, and archived. The original is implemented in allCLEAR. If a file containing the knowledge already exists, it is checked out of the CM library and updated. The use of the tool's search function to locate the headers of various troubleshooting paths represented in a particular allCLEAR file enables the maintainer to locate the section which must be updated. After Quality Assurance (QA) of the diagnostic trees has been accomplished, the file containing the updated knowledge is checked into the CM library. Older versions of the file containing the knowledge are archived. The archive capability is a feature of PVCS.

Future changes and diagnostic trees will be in the form of allCLEAR files created by the domain expert and saved on diskette. These files will facilitate the establishment of a paperless process of knowledge acquisition and representation. Though this tool is particularly useful for procedural knowledge, the author recommends its use in rule based applications also be explored.

d. Changes to the Knowledge Base

The change process outlined in this subsection assumes the use of allCLEAR. To implement changes to the knowledge base, the CLA should configure PVCS with allCLEAR as the text editor. Subsection E is an example of a change process that incorporates changes to the MK92 MAES knowledge base.

As with changes to the MK92 MAES program, decisions to change knowledge will be made by the CCB. The decision to incorporate changes will be through a by-negation process. That is, NPS should incorporate changes provided by

NSWC-PHD into the knowledge base unless there is a very good reason for not doing so. It should be assumed NSWC-PHD is sending change requests to knowledge that are considered necessary for implementation. This change request should be presented to the CCB with the understanding that changes will be made unless a reason can be provided as to why they should not be incorporated in the knowledge base. The decision by the CCB to approve a change to the knowledge base triggers a change request for the implementation of the knowledge in the expert system.

Using allCLEAR, implementation of approved changes to the knowledge base would be as follows: The appropriate allCLEAR file from the PVCS archives would be checked-out using the "Get" icon in the PVCS main window. By selecting the "Edit" icon, allCLEAR will be started, and the programmer may make his/her changes. Upon completion of the editing session, the programmer should check-in the revised file using the "Put" icon in the PVCS main window.

6. Configuration Status Accounting recommendations for the MK92 MAES

a. Access Control List

The Access Control Report is a report that provides a listing of the user groups, users, and their assigned access privileges. The report is provided to the project manager and student project leader by the CLA. The Access control list should be produced quarterly.

b. Configuration Item Report

The Configuration Item Report is a report listing all items under configuration management. It should be generated quarterly upon rotation of CM manager and upon turnover of the CLA. Generation of the report is the responsibility of the configuration library administrator.

c. Change Status Report

A change status report should be produced that gives the latest status of approved change requests. Its availability will depend upon the features of the tool purchased for tracking changes.

The change status report should be provided to each member of the CCB when it meets. Additionally, all project team members attending project meetings should be given a copy of the change status report. Change status reports should be part of the turnover given to faculty members by project team members as they transfer. When project meetings are being held, the frequency of change status reporting shall be determined by the project manager. Generation of the change status report is the responsibility of the configuration manager.

d. PVCS Reports

The following is a list of the PVCS reports. These reports can be produced on demand by any developer with the appropriate access.

- **Difference Report.** The Difference Report displays the modifications made between two revisions or files. The generation of difference reports between two binary files is not very useful.
- **Archive Report.** The Archive Report is used to provide information on archives and the revisions they contain.
- **Journal Report.** The journal report is used to document the activity of an archive. The basic foundation of a journal report is the journal file. A journal file is a log maintained by PVCS of modifications made to project archives. This log records the name of the archive, the action used to modify the archive, who performed it, and when it was performed. The CLA can configure PVCS to keep a journal file

PVCS Reporter is an add-on tool that provides greater reporting and query capabilities for the MK92 MAES' archives. Through the use of PVCS Reporter, the development team can produce customized reports on archived information.

e. Version Description Document (VDD)

As described in Chapter III, VDDs are sent to customers along with MK92 MAES installation diskettes. Each time a new version is prepared, the VDD must be approved by the project manager. Generation of VDDs is the responsibility of the faculty member in charge of development or the student project leader, if applicable. The document is produced by the project member responsible for creating builds from the Installit scripts. An approved copy of the VDD for a particular release should be provided to the faculty project management and NSWC-PHD engineers.

7. Configuration Auditing Recommendations for the MK92 MAES

Audits should be conducted by project management and the student project leader. They should be scheduled at least quarterly and prior to the release of a new version of the MK92 MAES software. The types of audits include, Functional Configuration Audit, Physical Configuration Audit, and In-process Audit. Chapter III provides a detailed explanation of the types of configuration audits and the purpose of each.

E. AN EXAMPLE OF THE CHANGE PROCESS

To illustrate the recommendations presented in this chapter, an example is provided. In this example, a change is proposed to the knowledge represented in Figure 6-2 and Figure 6-3. The proposal is input to the CM process and the changes are made. The example assumes the domain knowledge has been represented using allCLEAR and implemented using Adept. It further assumes PVCS is used by NPS as the CM tool for the MK92 MAES project.

1. A Change to the MK92 FCS is Promulgated.

NSWC-PHD receives an update by the manufacturer which changes a reading which is taken on a component of the MK92 MOD 2 FCS. After review, the MK92 experts at NSWC-PHD come to the conclusion this change affects the MK92 MAES and

should be proposed for update of the expert system by developers at NPS. Figure 6-2 and Figure 6-3 represent the knowledge as it exists before the change.

A NSWC-PHD engineer represents the proposed change to the knowledge diagnostic tree diagram and mails it to the MK92 MAES developers at NPS along with an estimate of the impact of not implementing the change. This information is recorded on the latest version of the MK92 MAES Change Request Form. Upon receipt of the proposed change to the knowledge document, NPS developers assess the impact of its implementation on resources and the project's schedule. This should take into account not only its representation in allCLEAR but also its subsequent implementation in Adept.

At the weekly project meeting, the student project leader presents the proposed change to the development group, now exercising their role as the MK92 MAES CCB. After a description of the proposed change and a discussion of its impact on the software, project resources, and the schedule, a decision is reached regarding further action. The CCB agrees the change to the knowledge should be implemented in the version currently under development. They further agree, in addition to a change to the knowledge, the software should also be changed. Due dates are assigned for both the completion of the knowledge implementation and the change to the MK92 MAES software. The CCB has no further proposed changes to review, therefore "adjourns." The status of changes in progress are saved until later in the project meeting. The CCB is not the place to discuss the status of changes currently being implemented unless a change to the due date is required.

2. Implementing a Change to the Knowledge Base

The project manager, upon reviewing the recommendations of the CCB approves the change request. The change request is then filed in the Change Control Log Book in the "active" section.

Figure 6-4 is an example of the PVCS main window from which many of the following actions are initiated. The student project leader has assigned the change to the student whose responsibility is to make changes to the calibration module. The student

takes the active change request from the log book and launches PVCS. The programmer selects the "Open" icon from the PVCS main window and the project entitled MK92 MAES. He then selects the "Wkngset" icon and chooses "MK92 MAES Knowledge Base."

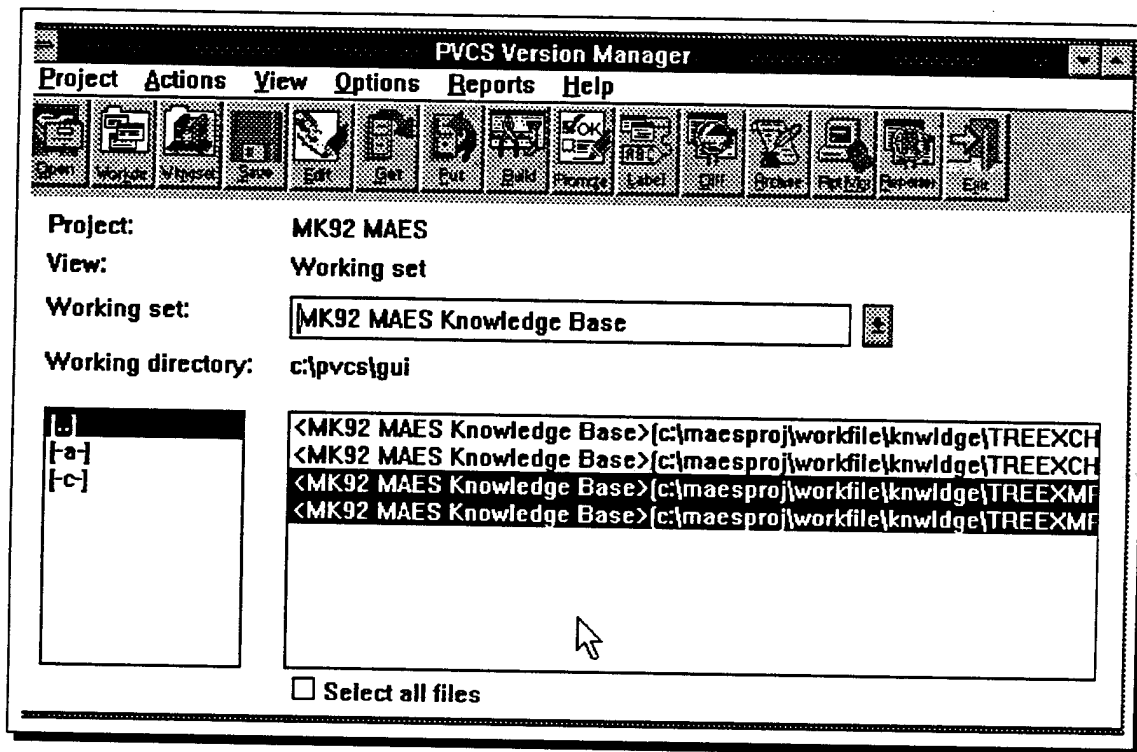


Figure 6-4. PVCS Main Window for the MK92 MAES.

To make a change, the programmer changes the working directory to the desired directory, chooses the desired files from the list in the main window, and selects the "Edit" icon. Because of the settings made by the CLA, PVCS automatically launches allCLEAR, checks out the latest version of each file to the programmer's working directory, and locks the files so others cannot make revisions to them.

Using allCLEAR, the programmer adds the changes to the knowledge highlighted in Figure 6-5. The resulting graphical representation of the knowledge is shown on Figure 6-6.

The programmer, upon completion, checks-in the revised file by exiting allCLEAR, choosing the file that has been checked out from the file list in the PVCS main window, and selecting the "Put" icon. The programmer completes the change process for a change to the knowledge base by completing the action taken portion of the change request and moving the document to the completed portion of the Change Control Log Book.

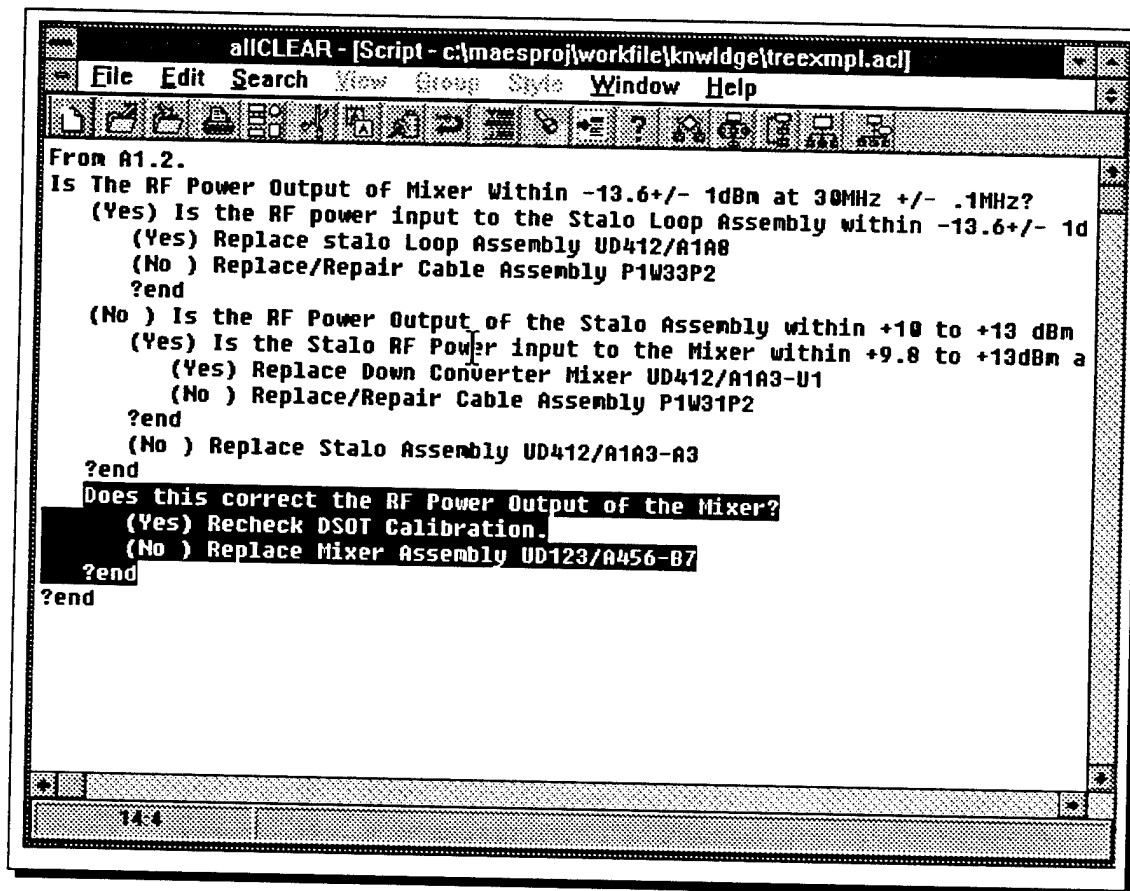


Figure 6-5. Change to MK92 MAES knowledge using allCLEAR.

To examine the differences between one version and another, the Difference Report capability is used. By selecting the "Dif" icon, the programmer can generate a difference report between two files or versions of a file. Appendix C provides examples of an Archive Report and a Difference Report as generated by PVCS.

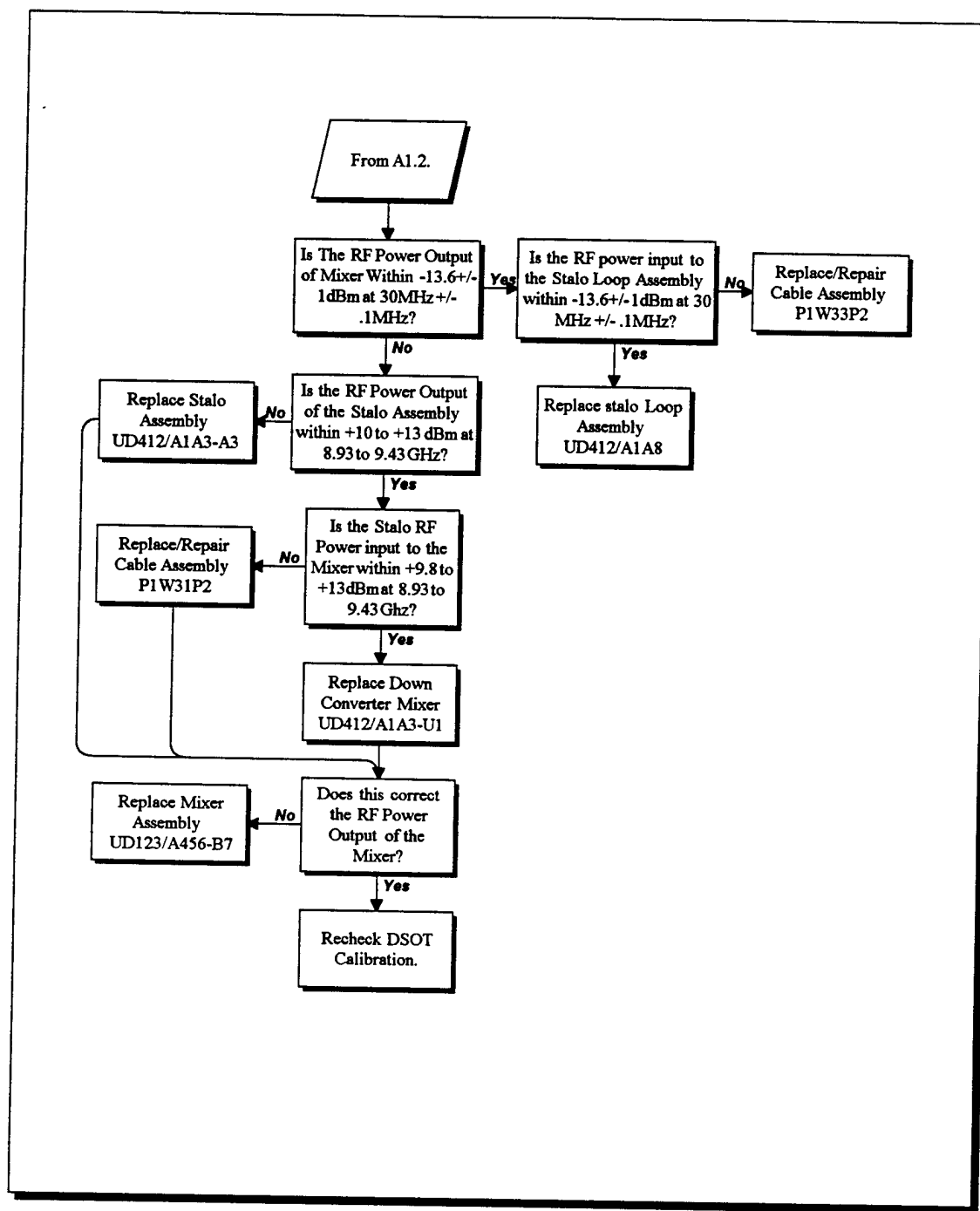


Figure 6-6. Graphical representation of a change to MK92 MAES knowledge using allCLEAR.

Changes to the MK92 MAES implemented modules are conducted the same way as those done on the knowledge. However, instead of launching allCLEAR and checking-out files using the "Edit" icon, the programmer will have to check out the desired files using the "Get" icon. Additionally, the programmer will have to minimize PVCS and start Adept outside of PVCS. Upon completion of desired changes, the developer should save the file, exit Adept, maximize PVCS, and check-in the revised file in the same manner as was done for the knowledge document. Unfortunately, because Adept files are binary files, a Differences Report cannot be generated.

Once the changes have been completed, they can be migrated to the testing archive for IV&V. To expedite the change process, the CLA or programmer should create custom workfiles that group knowledge modules with their associated MK92 MAES modules. This would allow the programmer to remain within the same working set for the entire change process. This also facilitates the IV&V process by grouping domain knowledge with its associated implementation.

F. CM RECOMMENDATIONS FOR AN EXPERT SYSTEM DEVELOPMENT CENTER

This section presents recommended modifications to the MK92 MAES' CM plan for its application to a small expert system development center.

1. Assumptions

Before one can make recommendations for scaling the MK92 MAES' CM plan to an expert development center, it is necessary to make several assumptions.

- The development team consists of three to five full time developers. For a five member development team, three will be programmers. The remaining two developers will make up the IV&V team.
- Project management will consist of two faculty members.
- Graduate students will be involved in thesis related research.
- The development team can be expected to be involved in the development of two expert system projects at any one time.

- The expert systems under development are diagnostic systems similar to MK92 MAES and will use the procedural network paradigm.

2. Recommendations for CM at an Expert System Development Center

Many of the recommendations made for the MK92 MAES project team will apply to the expert system development center. The number of personnel working for the development center and its expert system development process will be similar to the MK92 MAES project. The difference lies mainly in the fact that the development center is an environment in which greater structure and continuity will be maintained than the MK92 MAES development environment. Additionally, full time personnel will be available to offset the increased workload.

The PM will need to ensure the CM plan is adapted to new requirements such as the use of different development tools or the introduction of multimedia technology.

Although students are going to be involved with the project, it is recommended the CM tasks be accomplished by the full-time developers. By doing this, continuity and corporate knowledge are maintained, thus minimizing the impact of transferring students on the project team.

The position of CM manager should be filled by the senior developer. The senior developer is in the best position to control the day to day management of the CM process. It is unlikely the CM tasks associated with a full-time development center of the size assumed here will require the establishment of a separate CM manager position.

The position of configuration library administrator should be assigned to one of the IV&V members of the design center. IV&V personnel focus on product integrity, and work independently of the developers. This allows the CLA to be independent of the programmers.

The development center should continue to use PVCS as an automated CM tool. PVCS can be expanded to accommodate the expanded CM needs of several small projects. It is therefore suitable to continue using PVCS.

The development team should all be on a local area network (LAN). This LAN should include the faculty member's computers. Through a local area network, the development team can access the configuration library, and management can produce reports on demand.

The change control board can be more formal. Because the development team will largely be made up of full-time personnel, separate meetings from the project progress meeting can be scheduled with minimal impact on the development effort.

Change frequencies may increase with full time development taking place; however the existence of full-time personnel and other unknown variables such as the complexity of the projects under development makes it impossible at this time to assess the impact of the increased workload on the CM process. As the development team becomes involved in multiple projects, the automation of the CM processes becomes increasingly important. Problem tracking software will be a necessity. Document management software would also be useful.

The principles of configuration identification won't change. The difference will be the number of items. A more formalized labeling method for documentation, similar to that recommended by Buckley (1993) and presented in Chapter III, may be necessary as the systems under development become more numerous and their complexity increases.

Configuration status accounting won't change. The same reports will be required. However, as the CM process matures, techniques such as the use of metrics may require the generation of additional reports.

Configuration auditing won't change. Just as with the MK92 MAES project, there will be a need for configuration auditing at the development center. However, as there will be several projects in progress at any one time, greater coordination will be necessary to ensure the accomplishment of audits does not impinge upon the development of other projects unnecessarily.

The development center's CM program will have to evolve to meet the specific challenges of the paradigm selected for implementation. This thesis assumes a procedural

network paradigm similar to that used in the MK92 will be used for every project. If the decision is made to use another expert system paradigm, then the CM plan should be evaluated and changed to meet the needs of the project.

The development center should have a general CM plan that addresses those aspects of CM that won't change from project to project. CM, however, must be tailored to the peculiarities of each specific project; therefore, a project specific addendum to the general CM plan must be drafted for each project.

VII. SUMMARY AND CONCLUSIONS

This chapter summarizes the findings of the research of this thesis on the design and implementation of a software configuration management plan for the MK92 MAES. It also makes recommendations for further research efforts on configuration of expert systems.

A. SUMMARY

The implementation of changes to software requires the dedication of time and project resources. A process in which changes are made on an ad hoc basis fails to take into account the economic justification of such changes. This lack of control over project resources can result in projects that are over budget and behind schedule. Configuration management (CM) is aimed at providing a control mechanism for the change process.

This section addresses the findings of this thesis with respect to the research questions introduced in Chapter 1.

1. How Can Configuration Management Concepts be Applied to the Implementation of an Expert System?

CM of an expert system implementation is similar to that of traditional software development. The concepts of configuration identification, change control, status accounting and auditing that apply to traditional software development apply equally to an expert system software implementation. The difference between the CM of expert systems and traditional software lies in the need to apply CM to the knowledge base. To apply CM to expert system software alone is to address only half the problem of ensuring the software integrity of an expert system. Through the application of CM to the expert system development process, controls are placed over the implementation of changes to the system. Decisions regarding resource allocation are made after evaluating the impact of proposed changes on the project schedule and the budget. In this way, changes are implemented to both the knowledge base and the expert system implementation in an orderly, controlled manner.

2. What are the Benefits of Implementing Configuration Management?

Configuration management provides control to the change process and maintains product integrity. Traceability is promoted between the expert system implementation and the knowledge base through the application of CM. Furthermore, through the reporting and auditing processes of status accounting and configuration audits, CM communicates the status of a software project to management. Finally, CM supports development and maintenance through version control and change control, thus giving programmers the ability to quickly track down problems with an application and correct them efficiently and effectively.

3. What Attributes of Expert Systems Present Unique Challenges, If Any, to Configuration Management?

One of the challenges of implementing CM is the lack of detailed specifications. The domain expert's knowledge serves as a surrogate for the specification of the expert system under development. The lack of specifications complicates the establishment of expert system baselines. This makes the traceability of an expert system traceability difficult.

Expert systems are typically developed using a prototyping implementation process. Additionally, an iterative process of eliciting and testing expert knowledge is used to develop the expert system's knowledge base. As a result, an expert system is subject to a higher frequency of changes than traditional software development efforts.

The expert system development environment can also add difficulty to the implementation of CM. Not only is there a possibility of the programmers being geographically distributed, the knowledge required for developing the system could be physically distributed as well. The communication problems created by this situation complicates the CM of the knowledge base and the implementation of the expert system. Chapter IV discusses the issues surrounding the application of CM to expert systems in greater detail.

4. What Issues Surround the Application of Configuration Management to Expert System Domain Knowledge?

To ensure product integrity of an expert system, CM must apply to both the knowledge base and the expert system implementation. Characteristics of the knowledge base that influence the need for its CM include its volatility, functional scope, and system size/complexity. The possibility of physically distributed knowledge complicates the CM problem. Finally, knowledge representation schemes may make it difficult to interpret the impact of changes to the system.

To be placed under configuration management, the expert's knowledge must be categorized and baselined. This thesis proposes a classification consisting of four categories of knowledge for CM purposes. These categories include working knowledge, captured knowledge, developmental knowledge, and product knowledge. Once the knowledge has been categorized, baselines are established. Knowledge baselines include the functional knowledge baseline, developmental knowledge baseline, and product knowledge baseline. Each knowledge baseline is a subset of a corresponding expert system software baseline. The expert system's code should be traceable to the knowledge baseline it implements.

Using the script-based flowcharting program, allCLEAR, a method for representing domain knowledge is presented that facilitates the application of CM to the knowledge base using automated tools. Chapter IV presents the issues surrounding the CM of expert system's knowledge base. Chapter VI includes an example of a change that is made to the MK92 MAES' knowledge base using allCLEAR and PVCS. Through the use of the automated tool PVCS' custom working sets, one can group the knowledge represented in allCLEAR with the corresponding modules coded in Adept. This facilitates the maintenance and independent verification and validation of the expert system.

5. How Can Automated Configuration Management Tools be Applied to a Configuration Management Program?

Automated configuration management tools ease the administrative burden of CM by providing features for archiving, change control, status accounting, and auditing. Additional features include those that provide for check-in/check-out, life-cycle modeling, change migration, and the ability to reconstruct any version of a configuration item.

In this thesis, the IEEE recommended practices for the evaluation of CASE tools is adapted for the evaluation of CM tools for their suitability for the MK92 MAES. A methodology for evaluation that included the formulation of evaluation criteria is also presented. Two CM tools were identified for evaluation, Intersolv's PVCS Version Manager (PVCS) and Softool's CCC/Manager. After applying the evaluation criteria to the two tools, the project team selected PVCS for application to the MK92 MAES. Although PVCS has a relatively steep learning curve, the degree to which PVCS can be customized makes it particularly suitable to the CM of expert systems. Both CM tools evaluated share a problem that is common to all automated CM tools in that neither is able to identify specific changes made to binary files, such as those created by the expert system development shell, Adept. Chapter V presents summaries of the CM tools' features, and the process used in their evaluation.

6. What Are the Implementation Issues Surrounding the Application of Configuration Management to the MK92 Maintenance Advisor Expert System?

Chapter VI presents detailed recommendations for the implementation of CM to the MK92 MAES. A CM plan for the MK92 MAES must take into account the work environment of the project. The autonomous nature of an academic environment presents a coordination problem that must be addressed by the CM plan. Additionally, the frequent turnover of students necessitates a process that is easy to learn and incorporates as much automation as possible. The CM process developed for the MK92 MAES must be flexible enough to process changes from many different sources in a timely manner.

Strict access control, version control, and check-in/check-out procedures are required for the MK92 MAES project. However, formalized, highly structured, Change Control Boards (CCBs) typical of organizations with strict hierarchies such as NASA are inappropriate for the academic setting of NPS and the nature of the project.

A CM process is recommended for the MK92 MAES project that incorporates PVCS. PVCS provides the necessary version control and check-in/check-out procedures required for the MK92 MAES. Working sets are tailored to enable maintainers to check-out Adept modules and their associated knowledge documents, represented in allCLEAR, for simultaneous update.

Problems or recommendations for enhancement are recorded on a Change Request Form. This form is then presented to the MK92 MAES project's equivalent of CCB along with an analysis of the proposed change's impact upon project resources and the product under development. The CCB for the MK92 should meet as an integral part of the development team's scheduled project meetings. This approach is best suited to the project environment of the MK92 MAES.

B. RECOMMENDATIONS

The following subsections provides recommendations for future enhancements to the MK92 MAES' CM process and directions for future research in the development and refinement of a CM methodology for expert systems.

1. Recommendations for Future MK92 MAES CM Initiatives

a. Network the MK92 MAES Project's Computers.

To support check-in/check-out and other CM functions in a distributed development environment, it is recommended MK92 MAES developer's computers be incorporated into a local area network (LAN). The establishment of a LAN will facilitate the implementation of changes to configuration items downloaded from the configuration library.

b. Identify CM Support Tools That Further Automate the MK92 MAES CM Process.

Further research into the incorporation of CM support products such as documentation tracking tools, build software, and problem tracking tools in an expert system development environment is highly recommended. As the MK92 MAES is being tested, problems will be identified. The ability to automate the problem tracking process will be critical in keeping the CM of the MK92 MAES manageable for a small project team. The Change Control Log Book should be replaced with a problem/change tracking tool.

c. Send Prospective Configuration Library Administrators (CLA) to PVCS Training

The person assigned to be the CLA should attend Intersolv's PVCS Version Manager training course. This will reduce the PVCS learning curve, which as Chapter V points out, can be steep.

d. Increase Secondary Storage for MK92 MAES Computers

PVCS archives of binary files are compressed, however, as the MK92 MAES project matures and changes accumulate, additional storage space will be necessary. The hard disk drives of the project's computers are nearing capacity. An additional hard drive will be required to provide ample storage space for configuration items.

e. Train NSWC-PHD Engineers to Use allCLEAR for Knowledge Representation.

NSWC-PHD engineers manually draw diagnostic trees that constitute the knowledge base. Once the diagnostic trees are received by NPS, they are entered into allCLEAR to produce a knowledge base that can be placed under CM using the features of PVCS. To streamline this process, the possibility of training the NSWC-PHD engineers to use allCLEAR should be considered. This may require the establishment of a greater degree of CM at NSWC-PHD to control changes to the knowledge base as it evolves.

The difficulty of establishing a CM process that adequately controls changes made both at NPS and NSWC-PHD may outweigh any perceived benefits that come from "streamlining" the representation of the knowledge base.

One solution may be to have the NSWC-PHD engineers use allCLEAR for knowledge representation, but still maintain CM control at NPS. Under this process, AllCLEAR disks sent by NSWC-PHD could be placed under CM just as it is currently done. The use of allCLEAR by NSWC would eliminate the extra step of translating diagnostic trees into allCLEAR scripts.

2. Recommendations for Further Research

Research should be undertaken to develop configuration management tools which can manage changes to programs written in visual programming languages. As was demonstrated in Chapter V, current technology does not adequately address many of the latest trends in software development. Those that track changes to visual languages require the visual language be saved in a text format before useful CM reports can be generated and changes can be tracked. Development environments and CASE tools should incorporate software engineering disciplines that promote the development of maintainable code .

One problem with the reporting capabilities of PVCS, and CM tools in general, is its inability to identify changes made to binary files. As 4GLs and visual languages grow to dominate the marketplace, improvements in either 4GL development tools or CM support tools will be necessary if CM is to be adaptable to future development environments. Research that addresses the challenges of applying CM to 4GLs should be undertaken.

Finally, it is recommended that research be conducted that examines the feasibility of establishing a center for the development of diagnostic expert systems for the Navy. Although there are several efforts that are developing expert system technology, they do not seem to be coordinated nor organized. Useful output such as lessons learned, methodologies, and "best practices" are not resulting from these development efforts. The

result is a duplication of efforts in the areas of knowledge engineering, expert system development, expert system shell selection, project management techniques, and other aspects of expert system development. The establishment of a center would enable resources to be pooled and research efforts to be focused on solving problems associated with diagnostic expert system development. Further research is needed that would identify whether or not the concept of an expert system development center is economically justified.

C. CONCLUSION

The use of CM to control the changes to the software component of an expert system, in this author's opinion, is not nearly as significant as the application of CM to the domain knowledge from which the expert system is developed. The value of an expert system lies in the knowledge that is contained within it. Once the expert's knowledge is captured, represented, verified and validated, it can be implemented and adapted to suit almost any desired implementation technology. By developing and adopting a process that controls the changes independently to both the knowledge base and the expert system implementation, the goal of long-term maintainability is made more achievable.

If expert systems are to become a feasible alternative for the cost effective maintenance in the Navy, a maintenance oriented approach to their development must be taken. Configuration management is one approach that can be used to promote the implementation of maintainable diagnostic expert systems.

APPENDIX A. SOFTWARE CONFIGURATION MANAGEMENT STANDARDS

The standards and publications presented here are adapted from STSC (1994) and Buckley (1993).

A. DEPARTMENT OF DEFENSE STANDARDS AND PUBLICATIONS

The following is a list of DOD and military standards and publications of relevance to CM.

1. Department of Defense Standards

- DOD-STD-2168, Defense System Software Quality Program
- DOD 5010.19, Configuration Management

2. Military Standards

- MIL-STD-498, Software Development and Documentation
- MIL-STD-973, Configuration Management
- MIL-STD-1456, Configuration Management Plans

B. IEEE STANDARDS

- IEEE Std 610.12-1990, Glossary of Software Engineering Terminology
- IEEE Std 828-1990, Standard for Software Configuration Management Plans
- IEEE Std 1042-1986, Guide for Software Configuration Management

C. INTERNATIONAL STANDARDS ORGANIZATIONS (ISO) STANDARDS

- ISO/IEC JTC1/SC7/WG8/P.7.23, Software Configuration Management
- ISO 9000, Quality Assurance - Part 3, Software Configuration Management

D. ELECTRONIC INDUSTRY ASSOCIATION (EIA) PUBLICATIONS

- CMB6-3, Configuration Identification
- CMB6-4, Configuration Change Control
- CMB6-5, Textbook for Configuration Status Accounting
- CMB6-6, Textbook for Audits and Reviews

- CMB7-2, Guideline for Transitioning Configuration Management to an Automated Environment.

APPENDIX B. EXAMPLE OF A MK92 MAES CHANGE REQUEST FORM

MK92 MAES Change Request Form

Date	Site	Failed or Suspected Module	Version
------	------	----------------------------	---------

Description of Problem/Requested Change: (Please be as specific as possible. Attach additional pages if necessary)

Impact of Problem:

Recommended Action:

Expected Impact of Proposed Recommendation:

CCB Recommendation:	Date Reviewed:	Approved by:	Due Date:
---------------------	----------------	--------------	-----------

Action Taken: (Provide detailed description. Attach additional Sheets if necessary.)

Date Action Completed:	Action Completed by (Print):	Signature of Person Completing Action:
------------------------	------------------------------	--

APPENDIX C. PVCS REPORTS

A. EXAMPLE OF A PVCS ARCHIVE REPORT

Archive: C:\maesproj\archive\knwldge\treexmpl.stv
Workfile: TREEXMPL.STY
Archive created: 19 Dec 1994 18:12:22
Owner: unknown
Last trunk rev: 1.1
Locks:
Groups: Development : 1.1
Rev count: 2
Attributes:
WRITEPROTECT
CHECKLOCK
NOEXCLUSIVELOCK
EXPANDKEYWORDS
TRANSLATE
NOCOMPRESSDELTA
NOCOMPRESSWORKIMAGE
COMMENTPREFIX = " "
NEWLINE = "\r\n"
Version labels:
Description:

Rev 1.1
Checked in: 02 Jan 1995 11:51:44
Last modified: 02 Jan 1995 11:46:12
Author id: unknown lines deleted/added/moved: 17/28/1
This change to the calibration document accounts
for the possibility of a failed mixer assembly.

Rev 1.0
Checked in: 19 Dec 1994 18:12:58
Last modified: 13 Feb 1995 23:15:04
Author id: unknown lines deleted/added/moved: 0/0/0
Baseline of this file
=====

Archive: C:\maesproj\archive\knwldge\treexmpl.acv
Workfile: TREEXMPL.ACL
Archive created: 19 Dec 1994 18:12:22
Owner: unknown
Last trunk rev: 1.2
Locks:
Groups: Development : 1.2
Rev count: 3
Attributes:
WRITEPROTECT
CHECKLOCK
NOEXCLUSIVELOCK
EXPANDKEYWORDS
TRANSLATE
NOCOMPRESSDELTA
NOCOMPRESSWORKIMAGE
COMMENTPREFIX = " "
NEWLINE = "\r\n"
Version labels:
Description:

Rev 1.2

Checked in: 02 Jan 1995 11:51:44
Last modified: 02 Jan 1995 11:46:18
Author id: unknown lines deleted/added/moved: 0/4/0
This change to the calibration document accounts
for the possibility of a failed mixer assembly.

Rev 1.1

Checked in: 19 Dec 1994 19:43:22
Last modified: 19 Dec 1994 19:41:38
Author id: unknown lines deleted/added/moved: 4/0/0
Removed a statement that was in error

Rev 1.0

Checked in: 19 Dec 1994 18:12:56
Last modified: 26 Jan 1995 13:38:52
Author id: unknown lines deleted/added/moved: 0/0/0
Baseline of this file

B. EXAMPLE OF A PVCS DIFFERENCE REPORT

C:\maesproj\archive\knwldge\treexmpl.acv Rev 1.0 (26 Jan 1995 13:38:52)

C:\maesproj\archive\knwldge\treexmpl.acv Rev 1.2 (02 Jan 1995 11:46:18)

	12	12		(No) Replace Stalo Assembly UD412/A1A3-A3
	13	13		?end
+		14		Does this correct the RF Power Output of the Mixer?
+		15		(Yes) Recheck DSOT Calibration.
+		16		(No) Replace Mixer Assembly UD123/A456-B7
-	14			Does it work?
-	15			(Yes) Good you're done
-	16			(No) Keep going...
	17	17		?end
	18	18		?end

Note: The "+" symbols refer to those lines that have been added. The "-" symbols refer to those lines of code that have been deleted. The numbers in the left column refer to the line number of the code in the earlier revision, while the numbers in the right column refer to the location of the code in the latest version.

LIST OF REFERENCES

- Berlack, H.R., *Software Configuration Management*, John Wiley & Sons, Inc., 1992.
- Bersoff, E.H., "Elements of Software Configuration Management," *Tutorial: Software Engineering Project Management*, pp. 430-438, The Computer Society of the IEEE, 1984.
- Bielawski, L., and Lewand, R., "Maintenance Issues," *Expert Systems Development: Building PC-Based Applications* pp. 230-232 QED Information Services, Inc., 1988.
- Bounds, N.M., and Dart, S.A., *Configuration Management Plans: The Beginning to Your CM Solution*, Software Engineering Institute, 1993.
- Buckley, F.J., *Implementing Configuration Management*, IEEE Press, 1992.
- Curtis, B., "Maintaining the Software Process," *1992 IEEE Conference on Software Maintenance*, pp. 2-8, IEEE Computer Society Press, 1992.
- Eaton, D., "Commercial CM Tools," *Configuration Management Frequently Asked Questions*, Honeywell, 1994.
- CCC/Manager Primer, Softool Corporation, 1994a.
- CCC/Manager Release Notes Version 2.2, Softool Corporation, 1994b.
- CCC/Manager Order Form, Softool Corporation, 1994c.
- Deklava, S., "Delphi Study of Software Maintenance Problems," *1992 IEEE Conference on Software Maintenance*, pp. 10-17, IEEE Computer Society, 1992.
- Dills, K.R. & Tutt, T.F. *Verification and Validation of the MK92 Fire Control System Maintenance Advisor Expert System*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1994.
- Engineering Development Model (EDM): FCS MK 92 Maintenance Advisor Expert System*. Naval Surface Warfare Engineering Systems, Code 4W, 21 August 1992.
- Haga, W.J. and Lang, R.G., *Economic Analysis Procedures for ADP*, 3rd rev., Naval Postgraduate School, March 1992.

- Hicks, R.C., "A Composite Methodology for Low Maintenance Expert Systems Development," *Communications of the ACM*, v32, no. 3, 1989.
- Hull, L.G., and Kay, P., "Expert System Development Methodology and Management," *Proceedings of the IEEE/ACM International Conference on Developing and Managing Expert System*, pp. 38-44, IEEE Computer Society Press, 1991.
- IEEE Std 828-1983: IEEE Standard for Software Configuration Management*, IEEE Press, 1983.
- IEEE Std 1209-1992: IEEE Recommended Practice for the Evaluation and Selection of CASE Tools*, IEEE Computer Society, 1992.
- Interview with Ms. B. Kolkhorst, IBM, and the author, 8 November 1994.
- Jane's Fighting Ships 1993-94*, Jane's Information Group, 1994.
- Jones, C., "Inadequate Configuration Control and Project Repositories," *Assessment and Control of Software Risks* pp. 202-209, Yourdon Press, NJ, 1994.
- McCaffrey, M.J., "Maintenance of Expert Systems- The Upcoming Challenge," *Managing Expert Systems* pp. 262-284, IDEA Group Publishing, 1992.
- Lewis, C.D. *Development of a Maintenance Advisor Expert System for the MK 92 MOD 2 Fire Control System: FC-1 Designation-Time, FC-1 Track- Bearing, Elevation and Range, and FC-2 Track -Bearing, Elevation and Range*. Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.
- "Little Knowledge," *Computer World* v.28, pp. 86, 1994.
- "Military Standard Defense System Software Development," *DOD-STD-2167A*, Department of Defense (DOD), 1988.
- Powell, S.H. *Economic Analysis of the MK 92 MOD 2 Fire Control System Maintenance Advisor Expert System*. Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.
- Prerau, D.S., *Developing and Managing Expert*, Addison-Wesley Publishing Co., 1990.
- PVCS Version Manager Reference Guide Version 5.1*, Intersolv, 1993a.
- PVCS Graphical Interface Administrator Guide and Reference Version 5.1*, Intersolv, 1993b.

PVCS Graphical Interface Administrator Guide and Reference Version 5.1, Intersolv, 1993c.

Sacerdoti, E.D., "Managing Expert System Development," *AI Expert*, v.6, pp. 26-34
Miller Freeman Publications, 1991.

Smith, L.M. *Development of a Structured Design and Programming Methodology for Expert System Shells Utilizing a Visual Programming Language: Application of Structured Methodology to the MK92 Maintenance Advisor Expert System, Performance Module Prototype*. Master's Thesis, Naval Postgraduate School, Monterey, California, September 1994.

Software Configuration Management Technology Report, Software Technology Support Center, Hill Air Force Base, UT, 1994.

Telephone conversation between Henry Seto, Engineer, Naval Surface Warfare Center-Port Hueneme Division, and the author, 9 February 1995.

Tomayko, J.E., *Software Configuration Management: Curriculum Module SEI-CM-4-1.4*, Software Engineering Institute, 1990.

Torres, A., Miller, R., FCC Riley, Seto, H. *USS John A. Moore FCS MK 92 MOD 2 Groom, Port Visit Port Hueneme, CA 06 - 09 FEB 1995*, Naval Surface Warfare Center, Port Hueneme Division Memorandum, 1995.

USS Sides (FFG-14). (1995) *MK-92 MOD 2 FCS Maintenance Advisor Expert System (MAES) prototype evaluation*, Unclassified naval message, Date-time group 032252Z Feb 1995.

Walters, J.R., & Nielsen, N.R. *Crafting Knowledge Based Systems: Expert Systems Made Realistic*, John Wiley & Sons, Inc., 1988.

Wreden, N., "Configuration Management: Getting with the Program," *Beyond Computing*, pp. 49-51, International Business Machines, 1994.

Webster's New Collegiate Dictionary, p. 74, G. & C. Merriam Company, 1977.

INITIAL DISTRIBUTION LIST

	Number of Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Superintendent Attn: Library, Code 52 Naval Postgraduate School Monterey, California 93943-5101	2
3. Magdi N. Kamel, Code SM/Ka Department of Systems Management Naval Postgraduate School Monterey, California 93943-5002	2
4. Martin J. McCaffrey, Code SM/Mf Department of Systems Management Naval Postgraduate School Monterey, California 93943-5002	2
5. Commander, NSWC-PHD Attn: Henry Seto Code 4W32 4363 Missile Way Port Hueneme, California 93043-4307	2
6. Don Eaton, Code SM/Et Department of Systems Management Naval Postgraduate School Monterey, California 93943-5002	1
7. Bala Ramesh, Code SM/Ra Department of Systems Management Naval Postgraduate School Monterey, California 93943-5002	1
8. LT Paul G. Metzler, USN 6962 Berkshire Dr. Export, Pennsylvania 15632	1