Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1995-03

An examination of the Cosmos model for use in Department of Defense software development management

Drake, Steven G.

Monterey, California. Naval Postgraduate School

https://hdl.handle.net/10945/31539

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

DTIC
ELECTE
JUN 2 3 1995
S D
F

THESIS

AN EXAMINATION OF THE COSMOS MODEL FOR USE IN DEPARTMENT OF DEFENSE SOFTWARE DEVELOPMENT MANAGEMENT

by

Steven G. Drake

March, 1995

Principal Advisor: Keith Snider
Associate Advisor: Tarek Abdel-Hamid

Approved for public release; distribution is unlimited.

19950622 002

DTIC QUALITY INSPECTED 5

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	3. REPORT TYPE AND DATES COVERED
	March 1995	Master's Thesis

4. TITLE AND SUBTITLE An Examination of The Cosmos Model for use in Department of Defense Software Development Management	5. FUNDING NUMBERS
6. AUTHOR(S) Steven G. Drake	
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 200 words)*

Currently, the proper management of DoD software development projects is lacking. This is due, in large part, to the use of models of the software development process which neglect management aspects of the process. The Commonsense Management Model, "Cosmos", however, presents a complete view of this process by treating both its production and management facets. This model calls for a software development project manager to make three essential trade-offs. To make these essential trade-offs, a manager must consider the six principles of dealing with the dynamic complexity found in software development. Methods for dealing with these six principles can be found if the manager takes a three dimensional view of the software development process. Due to the conceptual nature of the Cosmos model, the model must first be grounded with "real world" examples before it can be effectively applied within DoD. To accomplish this, the Patriot software development management method is used to relate the concepts to specific examples for DoD use. By relating the concepts to examples, eight types of tools were found that could be used by future DoD software development projects to gain the benefit of a holistic view of the software development process presented by the Cosmos model. Specific recommendations are contained for inclusion in DoD policy with respect to software development management.

14. SUBJECT TERMS Software Development Management, Commonsense Management Model, Patriot Software Development Management, Patriot Software Development Process	15. NUMBER OF PAGES 94
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

AN EXAMINATION OF THE COSMOS MODEL FOR USE IN DEPARTMENT OF DEFENSE SOFTWARE DEVELOPMENT MANAGEMENT

by

Steven G. Drake
Captain, United States Army
B.S., University of Texas at El Paso, 1985

Accesion For
NTIS CRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification .....
By .....
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
March 1995

Author: [redacted]
Steven G. Drake

Approved by: [redacted]
Keith Snider, Principal Advisor

[redacted]
Tarek Abdel-Hamid, Associate Advisor

[redacted]
David Whipple, Chairman Department of Systems Management

ABSTRACT

Currently, the proper management of DoD software development projects is lacking. This is due, in large part, to the use of models of the software development process which neglect management aspects of the process. The Commonsense Management Model, "Cosmos," however, presents a complete view of theis process by treating both its production and management facets. This model calls for a software development project manager to make three essential trade-offs. To make these essential trade-offs, a manager must consider the six principles of dealing with the dynamic complexity found in software development. Methods for dealing with these six principles can be found if the manager takes a three deimensional view of the software development process. Due to the conceptual nature of the Cosmos model, the model must first be grounded with "real world" examples before it can be effectively applied within DoD. To accomplish this, the Patriot software development management method is used to relate the concepts to specific examples for DoD use. By relating the concepts to examples, eight types of tools were found that could be used by future DoD software development projects to gain the benefit of a holistic view of the software development process presented by the Cosmos model. Specific recommendations are contained for inclusion in DoD policy with respect to software development management.

TABLE OF CONTENTS

I. INTRODUCTION 1
A. BACKGROUND 1
B. THESIS OBJECTIVE 2
C PRIMARY AND SECONDARY THESIS QUESTIONS 3
D. RESEARCH SCOPE, LINITATIONS, AND ASSUMPTIONS 4
E. METHODOLOGY 4
F. DEFINITIONS AND ABBREVIATIONS 5
G. CHAPTER OUTLINE 5
II. SOFTWARE DEVELOPMENT MODELS 7
A. STATE OF WEAPON SYSTEMS 7
B. MANAGEMENT OF SOFTWARE DEVELOPMENT 8
C. CURRENT WIDELY USED SOFTWARE DEVELOPMENT PROCESS MODELS 10
D. THE COSMOS MODEL 16
1. Three Essential Trade-offs 16
2. Commonsense Principles 17
3. The Activity, Communication, and Infrastructure Framework 19
4. Use of the Cosmos Model 21
E. THE INTEGRATED SYSTEM DYNAMICS MODEL OF SOFTWARE DEVELOPMENT 24
F. MIL-STD-498 25
III. PATRIOT SOFTWARE DEVELOPMENT MANAGEMENT 27
A. THE PATRIOT EFFORT 27
B. SOFTWARE DEVELOPMENT MANAGEMENT METHOD 29
1. Public Laws and DoD Directives/Instructions 29
2. Software Development Management Overview 32
3. PDB Software Development Method 38
4. Configuration Management 42
5. Software Quality Assurance 44
IV. COSMOS IN DOD ANALYSIS 47
A. CONTEXT OF THE ANALYSIS 47
B. ACTIVITY DIMENSION 48
1. Separation of Concerns 48

2. Coevolution 50
3. Protoiteration 51
C. COMMUNICATION DIMENSION.....54
1. Inclusion 54
2. Reification 56
D. INFRASTRUCTURE DIMENSION.....58
1. Continual Improvement 58
E. INTERACTION OF THE THREE DIMENSIONS.....61
F. TOOLS FOR SUCCESSFUL DOD SOFTWARE DEVELOPMENT MANAGEMENT.....63
1. Engineering Service Contract 64
2. Post Deployment Build (PDB) Software Development Method 65
3. STEP Metrics 67
4. Risk Management Taxonomy 68
5. Software Development Library 69
6. Project Organizations 69
7. Personell Training Program 71
8. Role Maps 71
V. RECOMMENDATIONS AND AREAS FOR STUDY 73
A. SUMMARY.....73
B. RECOMMENDATIONS.....73
1. DoD Policy Recommendations 73
2. Recommendations for Patriot 77
C. AREAS RECOMMENDED FOR FURTHER STUDY.....78
LIST OF REFERENCES 81
INITIAL DISTRIBUTION LIST 85

I. INTRODUCTION

A. BACKGROUND

Software has become a key element in the design and development of sophisticated military weapon systems. This is primarily due to advances in micro-chip technology and programming techniques which have allowed processes to be accomplished either for the first time or less expensively when compared with accomplishing these events with hardware items. Software is now critical in giving today's modern weapon systems the ability to carry out crucial mission functions [Ref. 46:p. 2]. While software is considered an integral part of a modern weapon systems and allows for mission functions to occur, it is also known to be an expensive and technically difficult component that is estimated to range in cost from $24 billion to $32 billion annually - about 8 to 11 percent of the total National Defense budget [Ref. 46:p. 1]. If the current trends continue, this amount is expected to rise.

Although a great deal of money has been spent on software development, many times software products still do not meet the user's needs, and overrun programmed costs and schedules. In response to this, the Government along with major defense contractors have realized that successful and cost effective software development requires management. [Ref. 1:p. 1-1]

In an attempt to manage effectively, industry has turned to the engineering process of modeling and metrics. Using a quantitative approach for modeling the software development process is supported by experts in the field of software development. [Ref. 16,18,22] They believe that this is the direction in which software development must move to become a true engineering discipline and to satisfy the future demands for software development. Specifically, not only do these experts believe that we need models of the development process, but they also believe that we need measures of its characteristics and practical mechanisms for obtaining those

measures. Only then can we effectively manage the development process.

While the need for modeling of the software development process is recognized as being important, current models are inadequate because they only treat one side of the process. Although a great deal of attention has been given in the literature to the advancement of the technical side of the process, little has been given to the management side. [Ref. 16,21] What is needed are models that take a "holistic view" of the software development process [Ref. 12,16,17].

This means that the model of the software development process must consider both the management as well as the production functions of software development [Ref. 16]. These two facets of the software development process are recognized as distinct yet interrelated views that must be considered for successful software development [Ref. 15,16,21,24].

Currently, several software development models exist throughout commercial industry. Most, however, do not provide a full view of the software development process. Even so, these models have become the basis for many Government and industry standards. [Ref. 20,21,24,26,27]

B. THESIS OBJECTIVE

With the realization by both the Government and private industry that software is essential and a major cost driver for all new critical weapon system programs, the objective of this thesis is to analyze a software development management model presented in the literature and illustrate how it can be applied to actual large Department of Defense (DoD) software intensive weapon systems. The benefits of this analysis are twofold:

1. The identification of a new development management model that might benefit other programs in which software development is a large part; and
2. The identification of possible limits for the

effective application of the analyzed software development management model.

The Commonsense Management Model, "Cosmos," is a highly conceptual model that provides a holistic view of the software development process. To gain benefit from the model for the use in DoD, it must be grounded in practice. Therefore, the analysis of the Cosmos model for software development management will be accomplished by relating the functions identified in the model with the methods for software development management used by the Patriot missile system program office and its prime contractor, Raytheon. The reasons for use of the Patriot missile system as a case study in the analysis of the Cosmos model are twofold. The first reason is that the weapon system is a very large DoD procurement and is generally considered to be a highly software intensive, complex, and successful Army weapon systems program. The second reason stems from a professional interest in the weapon system that the researcher has developed through ten years of serving as a Patriot missile system officer.

C. PRIMARY AND SECONDARY THESIS QUESTIONS

To effectively accomplish the above thesis objective, the following research questions are asked:

A. Primary Research Question: How does the Cosmos model present a holistic view of the software development process, and how can it be used as a basis for military software development management?

B. Subsidiary Questions:

1. What are methods for software development management described by the Cosmos model?

2. What methods of software development management are utilized by the Patriot Program Office and Raytheon?

3. How does the Patriot software development method illustrate the use of the Cosmos software development model?

4. What recommendations for changes to DoD policy/procedures can be made with respect to software development process management that could benefit future military software development management projects?

D. RESEARCH SCOPE, LIMITATIONS, AND ASSUMPTIONS

This thesis investigates the current state of software development management in the military and industry, and how the Cosmos model provides a holistic view of the software development management process.

This thesis also investigates the software development management methods used for Patriot software. Additionally, it illustrates how the Patriot method represents the use of the Cosmos model in a DoD environment. This thesis also determines which types of tools, used by the Patriot software development management method, could be used by future military software development management projects.

This thesis does not look at DoD wide software development management methods, but limits the examples used for demonstrating the Cosmos model to those gained from the management of Patriot software development. Also, this thesis does not analyze to what extent the plans associated with the development of Patriot software are adhered to in practice, nor does it delve into the actual benefits and problems associated with the software code itself.

This thesis assumes that reader has an understanding of the DoD acquisition process and how it is used in the acquisition of software products.

E. METHODOLOGY

A comprehensive literature search was conducted to assess the current state of software development management in industry and DoD. The results of this search were used to establish the need, in industry and DoD, for a Cosmos type model. Next, the Cosmos software development management model is analyzed to determine how it represents a comprehensive model of the software development process.

Documents, through collection and review of Patriot documentation as well as through personal and telephonic interviews, were used to discern the Patriot software development management method. This investigation of the Patriot software development management method includes consideration for Public Laws and DoD policies governing military software acquisition and development. Following this, the Patriot software development method was related to the Cosmos model to show how the Patriot method demonstrates the Cosmos model in practice. Lastly, the analysis was extended to develop recommendations for DoD policy/procedural changes in software development management.

F. DEFINITIONS AND ABBREVIATIONS

The abbreviations found throughout this thesis are those that are common to the acquisition vernacular. However, due to the scope of the possible audience for this thesis, before an abbreviation is used it will first have its base word spelled out. When a definition of an abbreviation changes in this thesis, the abbreviation is redefined in terms of its new base word. Definitions for important concept words, key phases, and abbreviations along with their associated base words will be found in the glossary of the thesis.

G. CHAPTER OUTLINE

This thesis investigates the Cosmos software development management model and how it can be used for military software development management. The Patriot software development method is analyzed and used as a case study to demonstrate the model's application to military software development management.

Chapter I introduces the background and focus of the research. It discusses the current state of the use of software in modern weapon systems. It considers the need for management of the software development process and the requirement for its modeling.

Chapter II considers the current state of software

development management and presents several software development management models that are currently championed by industry. Additionally, it presents a complete description of the Cosmos software development management model.

Chapter III develops and presents the Patriot software development management method. It also considers the current Public Laws and DoD regulations that govern software acquisition in the military.

Chapter IV illustrates how the Cosmos model can be used in the DoD environment by providing examples of how the Patriot software development management method integrates concepts expressed in the Cosmos model. It discusses the types of tools used in the Patriot software development method which capture the concepts of the Cosmos model that could be used for the development of other large software intensive modern weapon systems.

Chapter V recommends policy changes for improving the effectiveness of the software development management methods used in DoD. It also provides areas of further study that have been brought to light during the course of this thesis.

II. SOFTWARE DEVELOPMENT MODELS

This chapter will review the current state of software development modeling for large weapon systems. By doing this it will make evident that there exists a need for change. This chapter will then present a possible solution.

A. STATE OF WEAPON SYSTEMS

Over the years military weapon systems have developed to an extreme point of complexity. Reasons for this can be found in examining the characteristics of modern development trends. Currently, trends show that user capability demands of weapon systems are so great that answers to these demands can no longer be obtained from simplistic solutions. Additionally, tighter schedules and smaller budgets for products cause cleaner simpler solutions to no longer be possible. In an attempt to deal with providing these complex solutions, contractors are turning more toward digitally based systems as a way to meet these complex weapon system demands. [Ref. 1:p. 1-2]

Although these digitally based systems provide flexibility and capability not possible in hardware intensive systems, they have not solved all of the problems associated with complex systems. Many software intensive weapons systems are still not delivered within schedule. Additionally, most do not meet acquisition cost ceilings or performance needs of the user upon initial delivery. [Ref. 1,2,3,4,5,6]

Although software cannot be blamed for all these problems with current systems, software is recognized as being on the critical path of system development and as such, has been found to be a major and many times the only contributor to the problems. "Software has become the Achilles heel of weapon systems." [Ref. 1:p. 2-7]

Contributing to this is the fact that far too many weapon system contractors are not fully qualified in the discipline of software engineering [Ref. 20:p. 279]. This

lack of qualification generally does not stem from a lack of understanding of the technical aspect of software engineering, but rather from not fully understanding the management aspect [Ref. 6,7,8,9,28]. What is needed are methods for better understanding the management of software development [Ref. 10,11,12,13,14].

B. MANAGEMENT OF SOFTWARE DEVELOPMENT

The current literature on the topic of software development offers clues to the answer for better understanding the management of software development. In particular, Reifer [Ref. 15:p. 2] writes that managing large software development projects suffers the same difficulties as managing other labor intensive activities:

> A large work force must be assembled and organized into teams. The engineering and management process needed to get the job done have to be solidified. Tool systems need to be acquired to support selected methods and to automate tedium. Requirements need to be specified along with customer's expectations. Plans need to be developed, and budgets and schedules need to be formalized. A variety of controls needs to be put into place as schedule and deliverables are defined. Staff must be acquired, trained, and motivated to perform agreed-upon tasks in a responsive manner. People need to collaborate, communicate, and be held accountable for results. Risk needs to be abated as managers respond , act, and perform their job, which is aimed at making things happen through the actions of others.

The focus of software development management then can be summarized as the art of planning, controlling, staffing, organizing, directing and integrating the efforts of others [Ref. 15:p. 2,28].

Reifer states that software development management deals with the three P's: "people, process, and product" [Ref. 15:p.3]. The idea is that a manager must understand the people involved in the software development process to include both user and developer. Additionally, the manager

must understand the process by which the product is being developed as well as the product. The management effort is to "reconcile conflicts" among these three aspects of software development. [Ref. 15:p. 3] Specifically, this concept suggests that understanding the process alone will not be enough for success. The manager must consider how the people and the product affect the process. In other word, the process that one uses must be "humanized and productized" for it to work in practice. [Ref. 15:p. 3]

Considering this, software development can be viewed as a process that essentially has two facets. One facet of the process can be looked at as containing "management type functions" and the other can be looked at as containing "production type functions" [Ref. 16:p. 6]. Management type functions are considered those functions which deal with the concepts of planning, controlling, and staffing of the software development effort. Production type functions on the other hand include the concepts of designing, coding, reviewing, and testing the system software. [Ref. 16:p. 7]

Although this might appear to be a logical breakdown of the software development process, this concept of looking at management functions and production functions with equal weight appears to be a paradigm shift from the way the Government and the industry has understood software development in the past. [Ref. 6,8,9,10]

Since the 1970s, attempts have been made to bring discipline to software development through the use of engineering principles. Software engineering, as this new discipline is known, encompasses both the technical and management aspects of software development. [Ref. 7,17,18] The problem has been, however, that although a great deal of attention has been given in the literature to the advancement of the technical side of the process, little has been given to the management side. According to Merwin [Ref. 19:p. 20]:

> Programming disciplines such as top-down design, use of standardized high level programming languages, and program library support systems all

> contribute to production of reliable software on time, within budget. ... What is still missing is the overall management fabric which allows the senior project manager to understand and lead major data processing development efforts.

Evidence for this can be seen in the types of software development management process models that have been developed. One such model, the waterfall model, has become widely accepted and is strongly suggested in the DoD military standard on the topic of software development. These models, which are designed for the management of the software development process, deal principally with the facet of production type functions. Generally, only cursory attention is given to the management functions if any attention is given to them at all. [Ref. 16,21,23]

C. CURRENT WIDELY USED SOFTWARE DEVELOPMENT PROCESS MODELS

As stated earlier, in an attempt to bring discipline to the process of software development, the industry applied the concepts of engineering. One of the concepts of engineering is to model a process in terms that will allow the viewer to better understand its ramifications [Ref. 22:p. 90]. Holding true to this concept, experts in the field have proposed models for the process of software development. As a first attempt to manage and solve the problems associated with "ad hoc" software development management in the past, the "waterfall" model was created. [Ref. 21:p. 63]

The waterfall model, shown in Figure 1, is a stagewise model that looks at software development as a sequence of events that are accomplished in a linear fashion. These events are system feasibility, software plans and requirements, product design, detailed design, code, integration, implementation, and operations and maintenance. This model is also called a life cycle model because the sequence outlined in the model takes into consideration all of the activities that are involved from the conception to discontinuation of a software based system. [Ref. 21,25]

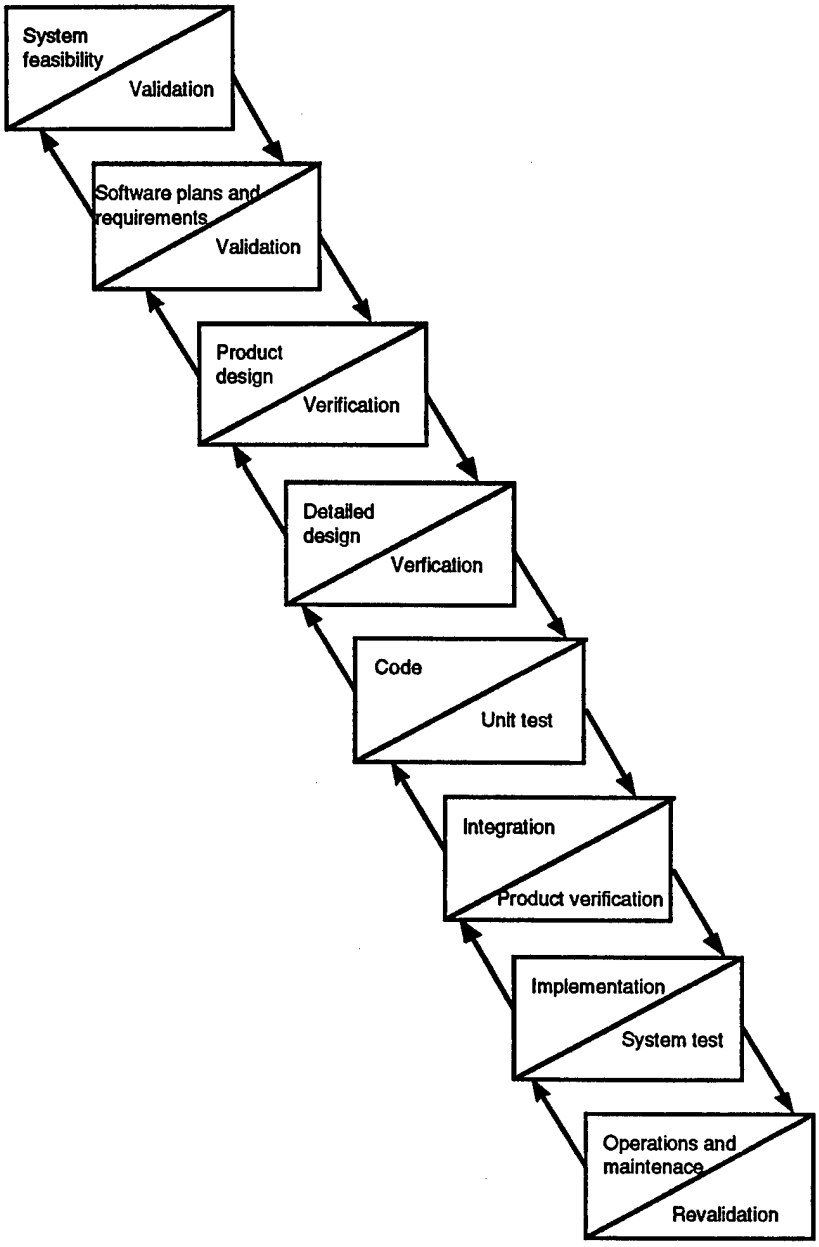

Figure 1. The Waterfall Model. After Ref.[21].

Aspects of the model that helped eliminate difficulties previously encountered in software projects are the recognition of feedback loops between stages, and guidelines for confining the feedback loops to successive stages to limit rework expense. Additionally, the concept of prototyping was introduced with this model as a parallel action accomplished early in the development cycle in order

to define requirements. [Ref.21:p. 63]

Although this model provides advances in the concept of discipline in software engineering, the focus of this model is primarily on production or activity functions of the software development process. While considering software design, coding, verification, and testing, the model does not consider the management type functions of planning for the overall software development effort, control, and staffing. [Ref. 24:p. 26]

Despite the model's lack of treatment of both facets of the software development process, it has become widely accepted as the backbone of most Government and industry software development standards. [Ref. 21,25] This is seen by its inclusion in the former DOD MIL-STD-2167A and now DOD MIL-STD-498 which govern software development in DoD [Ref. 26,27].

This lack of treatment of the management functions, however, was not the problem that led to the formulation of alternate process models. The focus of industry's complaints was that the waterfall model focused on the need for thoroughly elaborated documents as criteria for completion of the requirements and design phases. This was found, however, to be contradictory to the development of certain types of software. [Ref. 21:p. 63]

The problems with the waterfall model led to the formulation of the evolutionary model [Ref. 21:p. 64]. The evolutionary development model dealt with delivery of incremental capability to the user. The idea is that as the operational requirements of the user change, software would evolve to give the user the added capability. A benefit provided by the evolutionary model is that it brings early initial capability to users who do not know what they want but figure they will know it when they see it. Additionally, it provides a basis for additional product improvements. [Ref. 21:p. 64]

The issue with the evolutionary model with respect to this thesis is that, again as with the waterfall model, it

does not describe both facets of the software development process. Although the model introduces a longer term view of software development, thereby introducing elements of the management planning and staffing functions, the primary focus of this model is still on the technical activities of the software development process. [Ref. 24:p. 26]

As with the waterfall model, there were shortcomings with the evolutionary model recognized by industry. One problem with the evolutionary model is that it is difficult to distinguish it from the "ad hoc" form of software development because of the ill-formatted "spaghetti code" that it eventually produces. Also, it is based on the many times unrealistic assumption that the user's operational system will be able to accommodate unplanned evolution paths. [Ref. 21:p. 64]

Because of these shortcomings, the software development process evolved into the spiral model [Ref. 21:p. 65]. This model, shown in Figure 2, takes into consideration experience and refinements of the waterfall model as it has been used in large Government software projects. Also, not only can it accommodate previous models as special cases, it also gives guidance as to which combination of these models best fits a certain software project. This can in some degree be accomplished because the model is based on the progression through software development being tied to project risk. If the risk of proceeding to the next stage in software development is seen by the program manager as being low and acceptable, software development is allowed to continue. [Ref. 21:p. 66]

Essentially, the radial graphical construct of the model depicts that, as the project progresses, the risk analysis that is performed in each successive revolution will determine whether that project should stay on its current "evolutionary" path or whether another path should be taken up. For example, if at the risk analysis point in the software's development, the program manager is convinced that

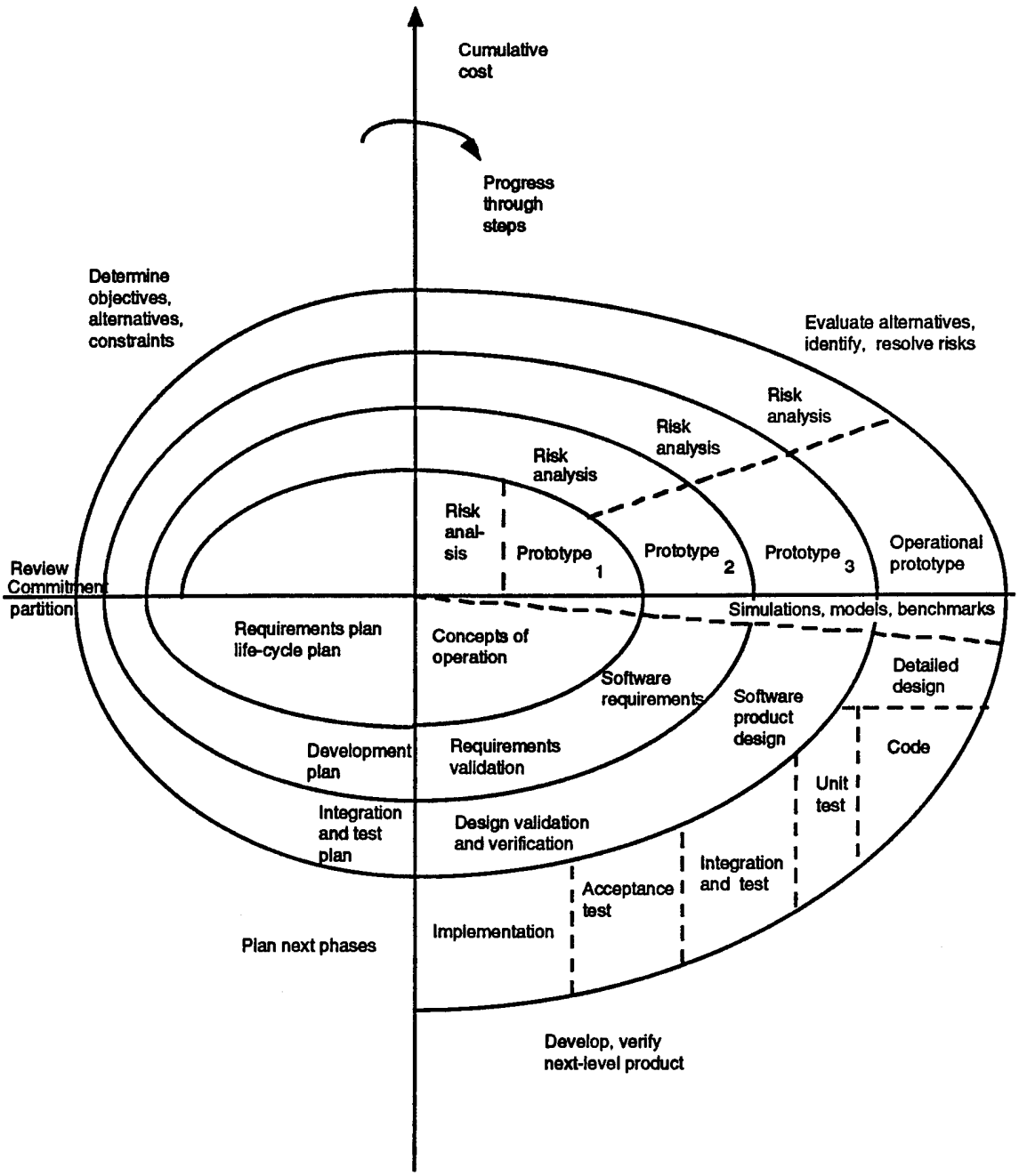

Figure 2. The Spiral Model. After Ref.[21].

requirements have been defined to the point that proceeding with the project in a waterfall model fashion would be of benefit, he or she can, at that point, embark on that path. If, however, further in the project the requirements change, the resulting risk analysis might emphasize an evolutionary model approach in the succeeding cycles of the spiral model. This flexibility allows the model to benefit from the good features of other models while the risk-driven approach avoids many of their difficulties. [Ref. 21:p. 67]

There are, however, problems with the spiral model that are recognized by industry. One difficulty is that the spiral model relies heavily on a program manager's or a software development agency's expertise in risk assessment. If the manager or his or her team are inexperienced in risk assessment, there is a probability that easily understood low risk elements of the project will be expressed in detail while little attention will be given to the poorly understood high risk areas. This could easily give the illusion that the project was on the path to success while actually it was heading for disaster. [Ref. 21:p. 71]

Another problem with the spiral model pertains to its lack of treatment of all the aspects of the management facet of the software development process. While risk management is well treated, the spiral model only implicitly considers other planning and controlling aspects of the process, it does not explicitly take them into consideration. [Ref. 24:p. 26] Therefore, the production side is well treated while the management facet languishes.

These models demonstrate the effort that has been placed on the modeling of the software development process in an attempt to gain an understanding of how better to manage it. However, they also demonstrate the lack of understanding of the dual nature of the software development process along with its requirement to be modeled. What these models do offer, however, is a basis from which other more comprehensive models can spring forth. One such model that has recently been put forth in the literature appears to take

into account both facets of the software development process. This model is called the Commonsense Management Model or the "Cosmos" model. [Ref. 24]

D. THE COSMOS MODEL

The Cosmos model treats the two complex facets of the software development process by looking at the issue from a three dimensional point of view. This three dimensional aspect is shown in Figure 3. By looking at the process from the Activity, Communication, and the Infrastructure dimensions, combined with understanding six principles that deal with dynamic complexity (that is, complexity that constantly changes as opposed to static complexity), the model allows the user to make three essential tradeoffs which are necessary for successfully managing software development. [Ref. 24]

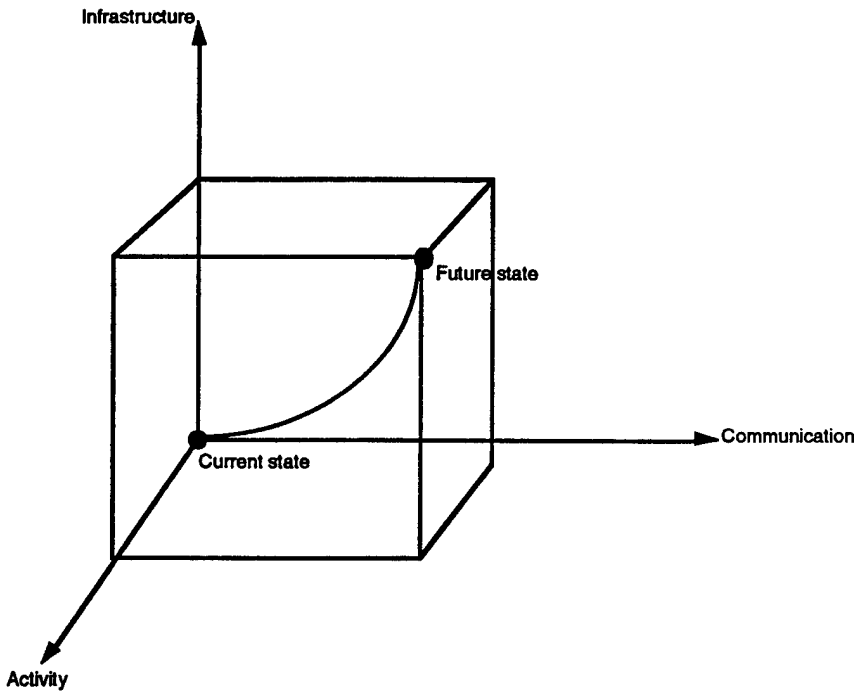

Figure 3. The Cosmos Model. After Ref. [21].

1. Three Essential Trade-offs

In order to effectively manage "large-scale, long-life projects" [Ref. 24:p. 23] the trade-offs of Flexibility

versus Stability, Modularity versus Interconnectivity, and Broad- versus Narrow-scope must be made.

The Flexibility versus Stability trade-off deals with the fact that large and lengthy software projects require schedule predictability and cost control, which is contrary to the unpredictable and intangible nature of complex problems. This trade-off is made by working in the Activity dimension and considering three of the six principles of dealing with dynamic complexity: separation of concerns, protoiteration, and coevolution.

The Modularity versus Interconnectivity trade-off deals with the fact that often the design of a large system is such that it is broken into modular subsystems. However, this concept conflicts with the idea that complex systems require a great deal of communication and interaction among all involved. This trade-off can be made by considering two of the six principles which apply to the communication dimension: inclusion and reification.

The Broad- versus Narrow-Scope (long-term versus short-term) objectives trade-off states that managers must balance cost and benefit optimization with predictability and control. This trade-off can be accomplished by creating methods that deal with the concepts found in the principle of continual improvement within the infrastructure dimension.

Understanding these tradeoffs is only part of the answer to effectively managing software development projects. The question of "How do we maintain effective balance among these difficult tradeoffs in the face of complex problems?" must be answered.

2. Commonsense Principles

To effectively balance these three tradeoffs, a manager must understand and implement the concepts of the six principles for tackling complex problems. The six principles mentioned above are: separation of concerns, coevolution, protoiteration, reification, inclusion, and continual improvement.

Separation of concerns deals with the concept of dividing and conquering. The focus of this principle is on subdivision and decomposition of a project to deal with its complexity. The manager must remember, however, that subdivision only makes sense if it does not add to the overall complexity. Additionally, the complexity of reintegration must also be understood.

The issue in the principle of coevolution is that many activities within the process must be developed in conjunction with one another. This concept is useful in deciding which activities might be developed concurrently.

Protoiteration means using prototyping in succession as a method of understanding the problem. The idea is that a single prototype cycle will not capture the right solution.

The principle of reification deals with clearly communicating information and rationale. The more clearly an objective is expressed, the more easily it can be understood by those who must accomplish it.

The principle of inclusion states that all individuals, groups, and stakeholders must be considered and allowed to participate in the projects development. The rationale for this is that such participation is beneficial in problem identification.

Continual improvement is the concept that all things can be improved upon. Protoiteration is used in this concept to ensure feedback from previous prototypes is introduced in the next iterations.

The ability of a software development manager to successfully manage a project lies with his or her ability to balance the three tradeoffs mentioned above. To accomplish this, the manager must understand the six principles of dealing with dynamic complexity. For example, to trade off flexibility and stability, managers of software projects that are ill-defined and poorly understood should be more flexible and adaptive until goals and means to reach these goals are well understood. Once goals are defined and means become available, managers can use more rigid advanced planning. To

facilitate this and assist in creating partial solutions, managers can use separation of concerns, protoiteration, and coevolution. These principles allow the manager to explore possible paths each time with incrementally small commitments.

To accomplish the Modularity versus Interconnectivity trade-off, two needs must be considered: a manager must isolate workers from extraneous distractions and ensure that they receive relevant information. Using the concept of separation of concern helps identify at various levels (process, project management, system architecture) how to break up the project into smaller modular structures that can be more easily dealt with by workers. The use of reification makes communication visible and explicit, and therefore ensures stakeholders in the development of the software communicate rationally. Inclusion deals with identifying who needs to be involved at what stage of the software development process so a manager can establish the proper communication channels.

Generally, long-term objectives are difficult to conceptualize, and are often confused with short-term objectives. By using the principle of continual improvement, managers can become aware of long-term objectives by creating infrastructures incrementally to support them. For example, if a corporate goal is to increase quality, managers might start toward that goal by implementing a process group to define, train, and measure quality. The principle of reification helps the manager to enforce his or her long-term goal because the rationale and background for the decision toward that goal are recorded and communicated appropriately.

To effectively visualize these six principles, the manager must look at his or her program from the following three dimensions.

3. The Activity, Communication, and Infrastructure Framework

The framework of the Cosmos model considers three

distinct interdependent dimensions of process modeling: activity, communication and infrastructure. This framework provides a method that allows the manager to understand how best to make the three essential tradeoffs mentioned above by bringing into focus the six principles of dynamic complexity.

The activity view deals with software project development with respect to what must be accomplished, how it must be accomplished, and when it must be accomplished. This is essential for dealing with the Flexibility versus Stability trade-off. Generally, this is the same view of the software development process that is described by models like the waterfall model.

The problem with the one-dimensional waterfall type model is that it leads to linear cause and effect type thinking. This sequential way of thinking by itself does not work for large software projects because it does not fully address the fact that many people and organizations must come together to solve large software development projects. Managers must realize that it takes time to bring people on board and train them. Additionally, they must understand that it takes years to develop systems and nurture management ability. On top of this, during this entire time, these processes continually interact. These concepts are not captured by such a model. Essentially, the one-dimensional activity view can handle detailed complexity, such as volumes of stepwise instructions as accomplished by the waterfall model; however, it is inadequate in handling dynamic complexity.

Addressing dynamic complexity begins by understanding the relationship among the various stakeholders, system components, and other elements of a project. By doing this, the manager is able to make the modularity versus interconnectivity trade-off. The catalyst for accomplishing this is the communication structure which models the communication channels among all stakeholders. This structure provides information like who should receive what information, in what format, and from whom.

The infrastructure view goes beyond the activity and communication views by taking into account what is needed to achieve project objectives. For example, a proper food supply is part of the logistical infrastructure to support military operational objectives. Process management is part of the infrastructure to build an organization that can develop a first-rate system.

While supporting communication and activities by providing structures that allow the accomplishment of these activities, infrastructure also evolves the process itself. With infrastructure, the feedback that is received about the process can then be used by management to make implied changes.

4. Use of the Cosmos Model

Important aspects of understanding the use of the Cosmos model are that the three perspectives of the model--activity, communication, and infrastructure--interact and assist in the evolution of each other. For example, if a manager cannot define the activity structure at a certain level because of the problem's complexity, he or she might observe the communication structure or infrastructure. By doing so, the manager might be able to clarify and create a certain activity structure and infrastructure from the communication view point.

Also key in understanding the Cosmos model is the idea of "a two process hierarchy". This hierarchy consists of the "control level" and the "execution level" [Ref. 24:p. 28]. Essentially this means that a manager will gain even greater benefit from the model by looking at it with respect to things that deal with the managing (control) of the process and project and also those things that deal with the technical (execution) aspects of the process.

Understanding this, the idea is to look at the software development project with respect to the three structures of the model (activity, communication, and infrastructure) in light of commonsense principles dealing with dynamic

complexity. For example, in looking at the activity structure which models the "how" of the software development process, a project manager would discern what activities at the control level and execution level need to be accomplished by looking at three of the six principles that apply to this model: separation of concerns, coevolution, and protoiteration.

Separation of concerns is obtained by breaking the project into subproblems or tasks that can be accomplished independently. This also applies when looking at the schedule of activities to be accomplished. The idea here is to decouple the schedule of events with the outputs these events create. By doing this, the events are freed to be accomplished the best way possible to meet output deadlines and not sequentially as perhaps the outputs are scheduled.

To apply the principle of coevolution, a manager must understand that all requirements for a software project cannot possibly be foreseen and will have to evolve as the "architecture and design" of the project evolve [Ref. 24:p. 30]. Additionally, coevolution is related to the separation of concerns in that a manager must ensure that all subproblems develop in a common vein so that they can be reassembled for later use.

Using the principle of protoiteration is required because rarely are the right solutions to dynamically complex problems found in the first try. Therefore, the structure of subproblems and schedules should be such that successive improvements can easily be made.

By using these three principles, often a manager will be able to have different activities ongoing concurrently. For example, the requirements for one software build can be accomplished while the design, coding and testing of other builds are also being accomplished. This, however, requires coordination and communication among the subproblems.

An aspect of the Cosmos model is that activity structures can be those of existing structures like the waterfall and spiral models. With well defined requirements,

the waterfall model could be used, while if risk management was a major consideration, the spiral model might be of greater benefit. This gives managers the flexibility to utilize different process models for different subproblems.

As stated earlier, the activity structure cannot be developed without consideration for its dependency on the other two structures of the Cosmos model. For instance, the use of separation of concerns, coevolution, and protoiteration are dependent upon the communication structure's modeling of "roles, interconnecting communication channels, and responsibilities and dependencies" [Ref. 24:p. 30].

The idea behind the communication structure is to make the act of communicating an explicit rather than an implicit function within a software development project. By doing this, the principle of inclusion is satisfied while the principle of reification acts as a guide to how the modeling of a role map (a map of who needs to talk to whom and how) is accomplished.

To develop a communications structure for a project, a manager must initially follow three steps in order. However, since communication needs will change as the project progresses, the communication structure may be updated in a sequence other than that presented below.

Initially, the manager must first determine who all the stakeholders for information of the project and subprojects are. This analysis includes determining who benefits from, who can constrain, and who will be affected by the structure. Secondly, the manager must determine "information about application domains, system use, tool use, relevant standards, and common practice," as well as each participant's responsibilities [Ref. 24:p. 31]. Lastly, the manager must determine how best to exchange information among and present information to various users.

To round out the complete understanding of the software development process, the manager must also consider project infrastructure. The rationale for this is its lack of

treatment by either of the other two structures.

Infrastructure captures the methods behind why events occur. Typically, this means discerning things other than delivered systems that support the meeting of objectives over the life process. The types of things included in this concept are project requirements and specifications, strategic objectives, communication structure, activity structure, application-domain modules, reuse strategy and support, life-process requirements, design and monitoring, and databases of test cases and scenarios collected from prototype or actual use. In addition to these tangible areas of infrastructure, intangible infrastructure such as feedback mechanisms for user inputs to fielded systems must also be developed.

With the current complexity of large scale systems and the increased reliance on digital solutions in the future, Cosmos offers a method for descriptively modeling the planning, process, and product of a software development project.

E. THE INTEGRATED SYSTEM DYNAMICS MODEL OF SOFTWARE DEVELOPMENT

The ultimate goal of modeling is to have a model define a process to such an extent that it transitions from being descriptive and becomes prescriptive in nature. [Ref. 16,22] With such a model, the user is able to ask "What if?" questions which allows him or her to discern the outcome of management decisions before they are implemented. One such model in the literature is the *The Integrated System Dynamics* model presented by Abdel-Hamid et al. Through its use of interrelated variables from the areas of the Human Resource Management Subsystem, Software Production Subsystem, Planning Subsystem, and the Control Subsystem, the model has been successful in predicting the staffing and schedule requirements of NASA's DE-A software development project. [Ref. 16] Although the System Dynamics model is currently only applicable to medium (16 to 64 thousand lines of code)

software projects, its significance to this thesis is that it demonstrates that a successful model of a software development process must include variables of the type that are found by looking at planning, process, and product; hence, the activity structure, communication structure, and infrastructure found in the Cosmos model. [Ref. 16,24,25]

F. MIL-STD-498

Realizing that current military standards do not adequately require a total view of the software development process, MIL-STD-498, *Software Development and Documentation,* was written in an attempt to gain a more holistic view [Ref. 27]. Although no longer required, due to the Secretary of Defense's order to discontinue use of all military standards, MIL-STD-498, the latest iteration of military policy governing DoD software development management, also speaks of many of the aspects in the Cosmos model. While explicitly stating that no specific software development management model is preferred, MIL-STD-498 does require that a software development process include fourteen major activities. It states, however, that these activities may overlap, be applied iteratively, applied differently to different elements of software, and need not be performed in any specific order. [Ref. 27] MIL-STD-498 emphasizes that "the development and recording of management and engineering information is an intrinsic part of the software development process..." [Ref. 27:p. 12]. Based on the way the activities are expressed, it is clear that MIL-STD-498 is designed to accomplish many of the concepts expressed by the three dimensions of the Cosmos model. Considering that this latest Government standard requires a software development manager to perform the types of activities expressed in the Cosmos model provides credence that the Cosmos model is on the right track.

This chapter has demonstrated how current methods of modeling the software development process fall short of encompassing both the management and the production facets of

the software development process. It also presented a recently described model that treats these two facets by viewing the software development process in terms of three dimensions: activity, communication, and infrastructure.

The next chapter describes the software development management method used by the Patriot missile system program office along with its prime contractor, Raytheon. This description will then be used in chapter four to illustrate how the Cosmos model can be used to provide for software development management in the DoD environment.

III. PATRIOT SOFTWARE DEVELOPMENT MANAGEMENT

The purpose of this chapter is to present the Patriot software development management method taking into consideration the Public Laws and DoD Directives/ Instructions that affect DoD software acquisition. The focus on the Patriot development method is essential for the analysis in Chapter IV of the Cosmos model and the Patriot software development method.

A. THE PATRIOT EFFORT

The Patriot missile weapon system was initially designed as replacement to the NIKE HERCULES weapon system to provide very low to very high altitude air defense to counter the air breathing threat (ABT) expected in the 1980s and 1990s. The mission of Patriot has since expanded to include anti-tactical ballistic missile (ATBM) defense. This increase in mission is provided to some degree by modifications made to the hardware, but primarily to the software of the system. [Ref. 29:p. 43,30:p. 1]

The Patriot missile weapon system is composed of several major essential pieces of equipment. At the Fire Unit (battery) level, the major end items are an Engagement Control Station (ECS) for command and control, an Antenna Mast Group (AMG) for digital voice and data communication, Launching Stations (LS) for providing fire power, and a multifunction phased array Radar Set (RS) for providing surveillance, target acquisition and track, and missile guidance. At the Battalion level, the Information Coordination Central (ICC) provides command and coordination functions for the Patriot Battalion Commander who is responsible for six Fire Units. [Ref. 31:p. B-1]

The Patriot system is the only ATBM weapon system in the U.S. military's inventory, and it has created a great demand for Patriot foreign military sales (FMS) since the Gulf War. Patriot is the largest fielding in Army Material Command (AMC) history. Specifically, the Patriot program is a $13

billion dollar program that has deployed 6576 missiles, over 80,000 major end items, over 60,000 publications and 250,000 spares, and tools. [Ref. 32]

To accomplish this massive program effort, the Patriot program's prime contractor, Raytheon, utilizes over 2000 contractors and vendors in 42 states. These contractors accomplish such tasks as the production of the Patriot missile, solid rocket motor, the launching stations, and power supplies. [Ref. 32]

Due to the nature of the risk involved in the creation and implementation of the technology required for the advances made within the Patriot system, a cost plus award fee contract (CPAF) is used to fund contractor work on the project [Ref. 33]. The statement of work (SOW) within the contract gives only the basic concept of the work to be accomplished while a technical directive order, known in the Patriot Program Office (PPO) as an Engineering Services Memorandum is used to define the specific work to be accomplished.

To ensure, however, that the prime contractor is making progress toward an end product, the PPO convenes an award fee analysis board to relook Raytheon's progress every six months. Specifically, the board decides on the evaluation that the Raytheon program office will receive and ways to expedite system modifications. Through this method of award, Raytheon receives a base four percent profit with a potential of between six and eight percent profit depending on the findings of the board. This process of award evaluation takes approximately ninety days to accomplish. [Ref. 34]

In addition to the CPAF contract used in Patriot program, an Engineering Services Program has also been implemented. Essentially, this program consists of a contract in which the Government purchases a number of man months from the prime contractor for use in the future. The Government may direct the contractor to use them as the Government sees the need. This method is extremely effective in allowing the Government the flexibility in implementing

quick changes to the system software to accommodate changes in requirements. The annual budget for the engineering services contract is $80 - $100 million of which $30 - $50 million goes to the maintaining and upgrading of software. [Ref. 33]

Specifically, this money is used to maintain and upgrade the 1,584,142 unique lines of code within the Patriot system [Ref. 35]. These lines of code are divided and reused among four separate areas of the Patriot system. Within the Fire Unit there are approximately 1,500,000 lines of code; in the Information Coordination Central (ICC) there are over 800,000 lines of software code; in the On-Line Tactical Training software there are approximately 400,000 lines of code; and making up the support software there are approximately 800,000 lines of code. These lines of code give the system functionality for the user to accomplish tasks ranging from defense planning, system set-up, and conducting air battle, to system maintenance, and air defense operation classroom training. [Ref. 32]

In order to create and upgrade these lines of software code, the Patriot Management Organization (PMO) has developed a method for managing effective software development. This software development management method is a function of several determining factors that range from Public Laws and DoD Directives to internal training and experience requirements.

B. SOFTWARE DEVELOPMENT MANAGEMENT METHOD

1. Public Laws and DoD Directives/Instructions

In order to minimize weapon system software support costs and to promote interoperability between the various systems, the Government has established Public Laws, Department of Defense Directives, and service-specific regulations that govern the software development and software development management processes. Although the majority of these directives and regulations were not initiated until the

late 1970s and later, many have since undergone several revisions and updates as the software development and its management process have become better understood. [Ref. 1]

Currently, one Public Law and several DoD Directives/ Instructions affect the software development management method of the Patriot weapon system. This law and these DoD Directives/Instructions cover topics such as the use of the Ada programming language, the use of prototyping, risk management, metrics, and the need for varied acquisition strategies. Examination of these documents reveals a desire for a commonsense approach to software acquisition and the idea of tailoring to meet the needs of a project. The result is that program managers are relatively free to create software development processes that meet the needs of their programs within the confines of the guidance. [Ref. 26,27,37,38,40,42]

In accordance with P.L. 102-396, DoD Directive 5000.1 and DoD Instruction 5000.2, all software for DoD usage will be programmed in the language Ada, unless specific service level waivers are given. Additionally, any projects currently in the production phase that have a change in software greater than 33% over the system life cycle are required to convert to Ada where it makes economic and technical sense to do so. [Ref. 36,37,38] Patriot meets this requirement since developers expect to change the system software between one to ten percent annually through the use of Post Deployment Build (PDB) upgrades. These changes are made to maintain and give the Patriot system the capability to effectively counter current and future air breathing and tactical ballistic missile threats. [Ref. 35]

In response to this requirement, the PPO has embarked on a parallel project to convert almost all of its 1.5 million plus lines of system software code, which are written in Jovial, Assembly, FORTRAN, and Microcode, into the Ada language. Because of the stringent timing requirements for accurate system operation, many time-critical weapons control component functions will remain in Assembly code to maintain

the required processing speed. [Ref. 35]

To accomplish the conversion, the PPO has decided against a risky and potentially catastrophic "all or nothing approach", and is instead proceeding with the conversion process incrementally, in order to mitigate as much of the risk as possible. The conversion process consists of using the Patriot system's Maintenance Control System (MCS), which is written in Jovial and Assembly language, as the initial conversion test case. Following this and incorporating the lessons learned in this thirteen month 60,000 standard lines of code (SLOC) effort, the plan is to program the entire Post Deployment Build-5 (PDB-5) (expected to be released in September 1998) in Ada. [Ref. 32]

Although the conversion program has been initiated and is well under way, several problems and risks had to be abated, and several risks still remain, which need to be addressed before the goal of PDB-5 written in Ada can be reached. One risk that was alleviated occurred prior to the initiation of the conversion program. To implement the conversion program, an Ada compiler for the Extended Weapons Control Computer (EWCC) had to first be created. Additionally, to automate this conversion process and mitigate problems with software coding error as well as delivery schedules, Raytheon developed a Jovial to Ada transformation tool. Also, due to differences in the way Jovial and Ada handle data, the system's data structure had to be redesigned. Lastly, specific functions of the EWCC system controller had to be modified to use Ada language outputs. [Ref. 32]

Assuaging these problems and risks was enough to get the program started, but to fully accomplish the goal of programming PDB-5 in Ada, other problems with their associated risks must still be mitigated. One risk that must be addressed is the completed development of a multiprocessor run time system. Also needed is unique compiler back ends for the Ada compiler. Other potential risks are the continued funding for the project which is expected to cost

$100-150 million over a four to seven year period, and the overall quality and coverage of the transformation tool. [Ref. 32]

Although the conversion process has several hurdles yet to overcome, the benefits for this conversion are considered by the PPO to be much greater. The benefits include the extensive Ada tool set which increases ease in software development, and the potentials for software reuse and maintenance due to the required structure in Ada coding. [Ref. 32]

MIL-STD-2167A is also a source of guidance that has had significant effect on the development management of Patriot software. Prior to the Secretary of Defense's (SECDEF) April 24, 1994 memorandum stating that military standards (MIL-STDs) were no longer to be used for product definition unless a service-level waiver was obtained, the software development team of the Patriot system began converting its software development process to meet MIL-STD-2167A (DoD Software Development and Acquisition) guidance. [Ref. 30,42] This MIL-STD, through the implementation of MIL-STD-1521B, requires that a contract data requirements list (CDRL) be provided to the product user in specific data item description (DID) formats. Also, MIL-STD-2167A suggests the waterfall model as a possible method for software development management. [Ref. 26:p. 10] Although this MIL-STD has recently been superseded both by MIL-STD-498, which provides similar guidance, and by the SECDEF's directive, currently the Patriot software development organization continues to use a tailored version of MIL-STD-2167A in its software development process. [Ref. 30:p. 1]

The following information is a summary of the software development management method for the Patriot system as outlined in the Patriot Software Development Plan. [Ref. 30]

2. Software Development Management Overview

The treatment of the Patriot software development process will begin with an overview of the Patriot program

management concept. This overview includes looking at organizational structure, schedule and milestones, formal reviews, risk management, software reuse, and personnel training. Following this, a more detailed look at the software development activity as well as software configuration management and quality assurance is given.

Within the Missile Systems Division (MSD) of the Raytheon Company exists the Patriot Program Management Organization (PMO). While the PMO has overall responsibility for program management, the MSD Missile Systems Laboratory (MSL) is responsible for the engineering tasks for both Patriot hardware and software.

Within MSL, Figure 4, several organizations are involved in the development of Patriot software. The following paragraphs name these organizations and briefly describe their functions.

A MSL Lead Engineer who is assigned for each separate Patriot project of software development has overall responsibility and will receive reports from the organizations listed below.

The Systems Design Laboratory (SDL) has the responsibility for the system requirements and many of the software requirements.

The Digital Systems Laboratory (DSL) has the responsibility for the digital design and subsystem diagnostics of the radar.

The Product Assurance Laboratory (PAL) contains the Software Quality Assurance (SQA) section which is responsible for software product evaluations. These product evaluations ensure that the software meets the requirements set forth by the Software Development Plan and other internal policy documents as well as those of the contract with respect to content and format. The PAL and thus the SQA are independent of the specific development organization they support. Therefore, although the SQA works closely with the software engineering effort, it is managed and reports through an independent chain.

The Configuration Management Laboratory (CML) contains the Software Configuration Management (SCM) organization which controls software configuration through configuration identification, change control, interface compatibility and status accounting. This organization works closely with the SQA organization and is under the control of the Software Lead Engineer.

The Software Laboratory (SWL) has the responsibility for software design to include related documentation, coding, unit testing, as well as software integrating, and validating. Additionally, SWL must analyze and assess all software requirements and create requirements for some Computer Software Configuration Items (CSCI).

To accomplish its tasks, the SWL, Figure 4, is further subdivided into functional departments. Like the MSL, these departments report to a lead engineer who in turn must report on a monthly basis to the SWL Manager, supporting departments and the PMO.

Within the SWL, the Application Software Department (ASD) has the responsibility for ensuring that build development and release of the software to the Configuration Management is accomplished.

The Systems Software Department (SSD) has the responsibility for developing the interface and validating Patriot system software. The validation of system software is accomplished by an independent testing section which is under separate management controls.

The Diagnostic/Test Software (DTS) Department has the responsibility for generating and delivering tactical software to the customer in addition to creating diagnostic and maintenance software.

The Missile Software Department has the job of creating embedded missile software.

The Software Development Center (SDC) has the responsibility of resourcing the software engineering effort.

The Software Engineering/Technology (SET) Department has the task of capturing data on, as well as improving the

software development process. This is accomplished by using the lead engineers' monthly program reviews to identify trends and the inputs from established Software Initiative Working Groups (SIWG). The SIWGs investigate to find ways for improvement in the areas of: Risk management, Project Management, Subcontractor Management, Associate Contractor Management, Requirements Management, Peer Reviews, Trusted Software, Process Measurement, Process Definition, Technology Management, Defect Analysis, and Software Quality Assurance.

To manage these departments and hence the software development effort, the Software Laboratory Lead Engineer controls schedule and milestone activities, electronically presenting monthly updates at the Software Lab Program Review.

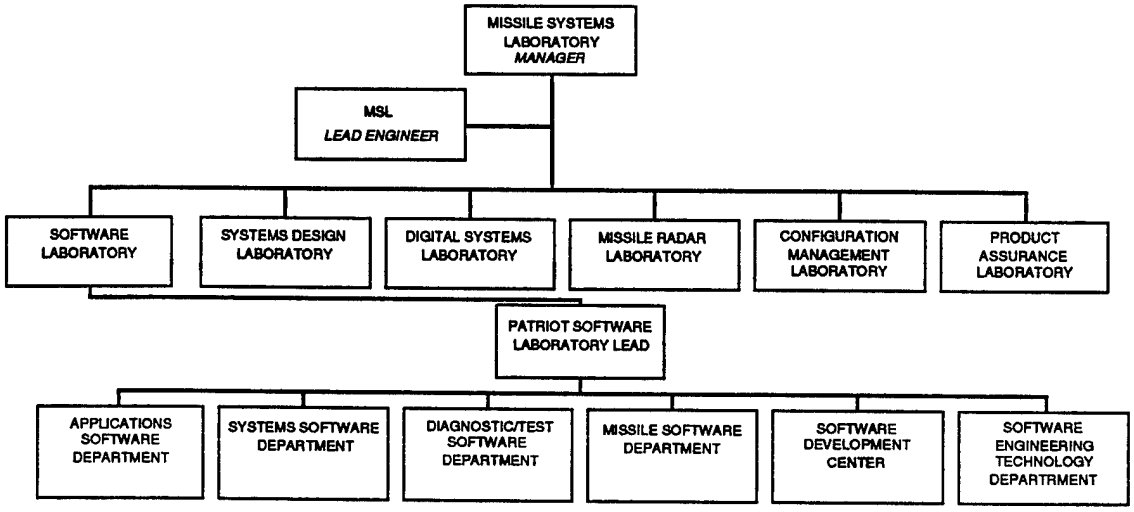

Figure 4. The Patriot Software Develop. Org. After Ref.[30].

The Software Lab Program Review, which at the start of a project is called the Software Lab Startup Review, has as members to be briefed: the Patriot Project Management Organization, the SWL Patriot Lead Engineers, Patriot Project CSCI heads, Software Engineering and Technology, and MSL Project Lead Engineers (including SQA). At these reviews, the SWL Lead Engineer, in addition to providing information on schedule and milestones, will also provide information on

program status (including prototyping activities, and incremental build activities), cost status, manpower analysis (including manpower profile), risk management plan, subcontract management issues, and metrics (including the set of STEP metrics found in DA PAM 73-1 [Ref. 43]).

Additional formal reviews, which are held to ensure the proper control and development of software, are the Test Readiness Review (TRR), the Flight Readiness Review, the MSL Reviews (which include Concept Review, Equipment Design Review, Pre-production Review, Product Readiness Review, and the Transition to Production Review), and the In-Process Review (IPR). These reviews, which are all internal except one, are designed to determine software integration readiness as well as overall software stability prior to start of Comprehensive Testing.

The IPR, which is chaired by the Government's Patriot Project Office (PPO), is attended by the user, the training and test community, and relevant contractors. This periodic review focuses on program status (including STEP metric results), problem issues requiring resolution, technical issues that need concurrence, and technical reviews as called for by MIL-STD-2167A with areas of interest specified by MIL-STD-1521B.

One important concept that is used to effectively manage Patriot project cost, schedule, and performance, and which is the essence of what is highlighted during the formal reviews, is risk management. Within Patriot software development, risk management is seen as a function of risk identification, analysis, mitigation, tracking, and control.

To accomplish risk identification, a "taxonomy of risks" [Ref. 30:p. 9-2] list exists which is updated as new risks that are not on the list are identified. The focus in risk identification is on finding the reason why a certain symptom exists. Included in risk identification is defining as clearly as possible what the risk is, the danger the risk could pose to the project, and what should be done to mitigate it.

Risk analysis consists of investigating the cost that the risk poses should it occur, and the likelihood that such a risk will occur. This enables the risks to be ranked as to the threat they pose to the project.

Risk mitigation occurs when the top five to seven listed risks have both strategies and closure criteria for their elimination. To assist in this process, a list of currently defined risk strategies exists and is updated as new strategies are developed. Upon selection of a mitigation strategy, a method for tracking the progress is identified. The progress is then reported each month at the SWL Program Review.

Risk tracking is accomplished by looking at the current, resolved, and new risks at each monthly SWL Program Review.

Risk control happens when the criteria for risk mitigation is met. Once this occurs, the risk is subsequently deleted from the active list. However, the history of the risk and lessons learned will be included in the Software Development File (SDF, a file holding documentation pertaining to a piece of software code) to be available for risk assessment on the possible reuse of the software on future software development projects.

The ability to reuse software and the lessons learned from its development is possible because of the Software Development Library that is maintained within the MSD. The Software Development Library is an electronic data base depository for the storage and controlled access to all software documentation, design artifacts, source code, object code, test specifications, test results, and project SDFs. Reuse of material held in this depository by projects is accomplished by choosing components from the Reusable Software Parts Catalog. If reuse is deemed appropriate by a project, items to be reused must be identified and evaluated during software detailed design. If on the other hand, a project identifies software components that have reuse potential, they are submitted to the reuse catalog and identified as such at monthly SWL Program Reviews.

To accomplish the above activities requires trained, qualified personnel. The need for proper education is recognized as an essential part of good software development. To this end, within each area of expertise with the software development process, required education and experience levels have been established for the various job titles. Additionally, each area has established training programs that ensure the proper development of its personnel to satisfy the project's requirements. For example, Software Engineering has an entry requirement of a minimum of a bachelors degree in an engineering, math, physics, or other related field. Upon entry, needed additional training requirements are established to ensure the person meets job requirements. These training activities may take place before or during software activities as necessary.

In addition to the areas discussed above, the Patriot software development process also includes the activities of software development, configuration management, and quality assurance. The following paragraphs provide detailed summaries of these activities.

3. PDB Software Development Method

The effects of MIL-STD-2167A can readily be seen in the sequence of the Patriot software development practice. Although the Patriot project uses a tailored version the MIL-STD, much of the terminology used and documentation created are in language and layout that make them MIL-STD-2167A compliant. For example, the terms Computer Software Unit is used to describe the smallest block of code that describes a complete function, and the term Computer Software Configuration Item is used to describe a complete software program (build) within a project. Figure 5 depicts the sequence of the software development method for the Patriot missile system. The paragraphs that follow offer explanations of this software development method.

Software requirements definition consists of the activities of prototyping, requirements generation,

requirements analysis, and requirements approval.

Prototyping allows for early discovery of requirements definition problems by taking into consideration human factors, timing and throughput needs, requirement completeness, and hardware interface issues. Plans for the prototyping activity are approved by the PMO and the Lead Engineer of the performing Laboratory. Within the SWL prototyping activities and their relationship with other activities are documented on the program schedules presented at the monthly Program Review.

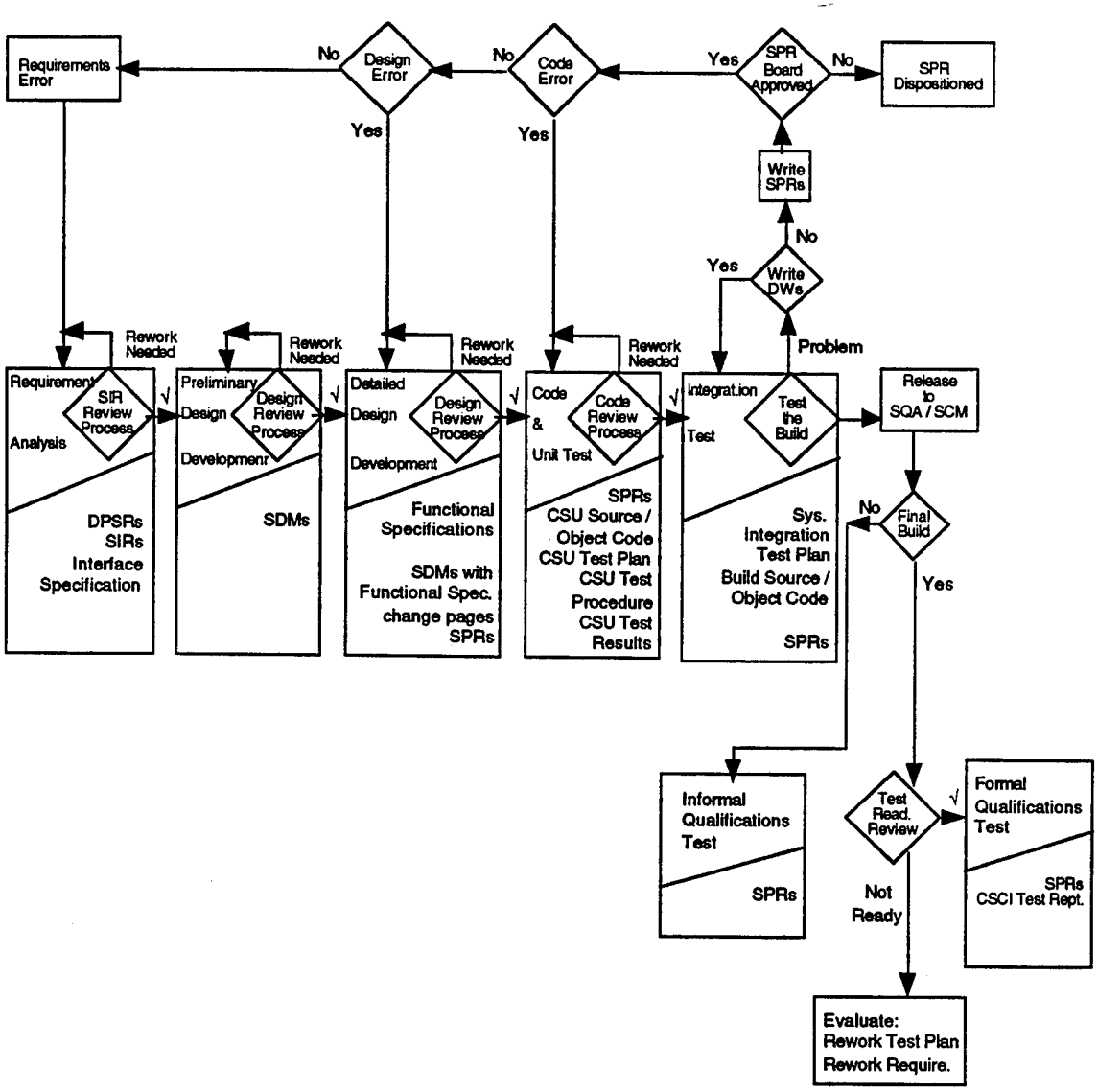

Figure 5. The PDB Software Development Method. After Ref.[30]

Following prototyping, the activity of requirements generation occurs. Essentially, requirements for each contract are documented and defined. These requirements generally take the form of Data Processing System Requirements (DPSR) which define the highest level of software system requirements and therefore define the baseline for the products the Software Lab produces. Any changes to the DPSR are requested by using a Software Investigation Request (SIR) form. Since Patriot is a deployed system with an established baseline, changes to the DPSR through the use of SIRs is the prime driver of requirements generation.

Once changes are requested by the Patriot PMO, the Systems Design Laboratory, the Software Laboratory, and others (such as the user), resulting SIRs are reviewed for approval and inclusion into future DPSRs. This is accomplished by an SIR Review Board which is chaired by the Patriot Software Development Manager (PMO). Once approved, new SIRs are assigned to the Software Laboratory for implementation.

Requirements analysis is accomplished by the Software Lab by analyzing the requests set forth in the SIRs. This analysis ensures that the specified requirements can be implemented and are testable. The analysis also ensures that the scope of the requirement is understood as it pertains to its implementation, support software, and testing. To satisfy the requirement, the SWL considers the possibility of reusing previous code as well as incorporating prototyping results.

Upon completion of the requirements analysis, the Lead Engineer schedules the requirements to go before a change review board for requirements approval. After all issues have been addressed and system impact understood, the requirement is approved for implementation.

Requirements implementation begins with the Software Design phase which includes both the preliminary and the detailed design. Specifically, as SIR documents affect

changes to the DPSRs, changes to the software design (Functional Specification) are accomplished by using a Software Design Memo (SDM). Within the SDM, information for both the preliminary design and detailed (functional specification) design is contained. Once the detailed design is completed, a review is held to ensure that the functional specification is detailed enough to demonstrate the connection between the DPSR and the code for each Computer Software Unit (CSU), and that the specification clearly describes all requirements and design information required to develop code.

Once the functional specification is approved and released, the coding process begins. Within Patriot, coding follows MIL-STD-2167A which, as mentioned earlier, defines code levels as Computer Software Configuration Items (CSCI), Computer Software Components (CSC), and CSUs. In Patriot, the CSCI maps to the term "build" while the CSC and CSU retain their standard definitions.

During code testing, should an error be found with the code or the functional specification, a Software Problem Report (SPR) is initiated. This report documents the problem and the corrective change. Once identified, the corrective change is made in the next coding cycle before the software is released.

Once coding is complete, CSU level reviews are held, and CSU testing has occurred, the CSU is then ready for incorporation into the baseline. This is accomplished through integration testing.

Software integration begins with a "Call", the request for the release of programs modified so that the modified CSUs can be incorporated into a build. Prior to being incorporated into a build, each program is reviewed by Software Integration. Programs are either approved or disapproved. If programs are disapproved, problems are documented on either a SPR or a Deviation Waiver (DW) and the program is returned to the author for repair. If the problem found is critical, an immediate octal patch is used to repair

the software and this is documented on a DW. If, however, the error is minor a SPR is written and a solution is implemented into the software at a later time. The review cycle continues until Software Integration approval is given. The review cycle also ensures notes made by reviewers are retained for later incorporation into the SDF.

Upon completion of the integration testing and the formal Test Readiness Review (TRR), the software is then prepared to begin the formal CSCI testing process. This process begins with the test development and continues into Informal Qualification Testing, and culminates in Formal Qualification Testing (FQT).

FQT is accomplished by an independent test organization dedicated to software testing and occurs at the Patriot Software Test Facility and the Missile Command (MICOM) Software Engineering Directorate Facility.

Once software is found to be correct it is then released to the user for implementation. Ensuring that only correct effective software is released to the user is a function of Software Configuration Organization.

4. Configuration Management

The major configuration management milestones associated with the software development process are the establishment of baselines per the software development schedule and the required reviews and audits scheduled at appropriate points during the program. To this end the tasks of software configuration identification, change control, interface compatibility, and status accounting have been assigned to the Software Configuration Management (SCM) section of the Configuration Management organization.

Configuration identification is accomplished when the documentation specified for a software project is released to the SCM. Acceptance of the requirements and the interface documentation by the SCM establishes the software requirements baseline for development configuration. This implies that design engineers understand the requirements

after the Requirements Review and the documentation has been placed in the software database. Identification of a configuration item will consist of the documentation being marked by project code, and document title, number, format and release date. When a revision is made to the specification, document, or code, it is treated as a complete reissue of the material and must follow the materials reidentification requirements mentioned above.

The concept of configuration control deals with the assurance that coordination of decision making functions occurs. Patriot implements this concept by ensuring revisions and problems are documented, submitted, reviewed, and approved/disapproved in accordance with configuration control procedures. These procedures deal with reporting documentation, review procedures, and storage, handling and release of software media.

The documents that are used to track the configuration process are the Engineering Release (ER) form, Software Problem Report (SPR), Software Investigation Request (SIR), Engineering Change Order (ECO), and the Deviation Waiver (DW). Since SPR, SIR and DW have been defined previously, the remaining documents will now be defined.

The ER form is used for proposing, transmitting, and recording the release actions of the Configuration Control Board (CCB) with respect to engineering documentation that is used to establish requirements, and design and code baselines.

Once software documentation or code has been released by the software CCB and is under SCM control, an ECO is required to propose, transmit, and record changes to the software CCB approved configuration. The ECO identifies by document number and revision level the baseline software and document to be changed, a complete description of the change, a justification for the change, and the approval signatures of the software CCB members.

The CCB has the responsibility of reviewing and evaluating all proposed engineering document releases and SPR

approved document changes to the software and associated documents. Although all changes to currently released software documentation and code are first documented on SPRs which are reviewed for approval/disapproval by the Software Review Board (SRB), these changes cannot be incorporated into the baseline until approved by the software CCB.

Once software media is released, the SCM requires two copies (one master and one working) to be placed in their control. Working copies will be stored in a central vault while all masters will be stored in an off-site storage facility.

Configuration Status Accounting (CSA) is accomplished through the use of an automated database. This database supplies the CSA personnel with data elements that are extracted from the ERs and ECOs that include document number, nomenclature, security classification, ER number, ECO number, and software CCB approval date. This data can then be manipulated into various report formats for use during the review processes.

While the job of the SCM organization is to control changes that affect the baseline software through communication among affected parties, it is the job of Software Quality Assurance to ensure that this specified communications continues to occur and that all requirements for effective development are met.

5. Software Quality Assurance

Although the responsibility for quality software lies with all who are involved with Patriot software development, the formal responsibility of quality assurance rests with the Software Quality Assurance organization. Members of this organization are classified as either Senior Engineer, Engineer, or Member of the Technical Staff depending on their experience with software quality assurance.

To accomplish software quality assurance, this organization, which is independent from the organizations developing the software, utilizes a separate set of databases

and utility programs for tracking the progress and ensuring the accuracy of software being developed. Through automated comparative analysis, members of the organization are able to assess differences between two files, discern the extent to which patch files are used, and to receive file listings of the CSUs contained on a source tape. During a review or inspection, should the SQA find a problem with the software development procedures, the Software Analysis Request (SAR) system will be used to identify, track, and close these issues.

Software quality assurance in the Patriot software development occurs during the entire software development cycle. For example, during the review boards, members of the quality assurance organization are present to ensure unbiased evaluation of the software's readiness to proceed to the next phase of development. Also, prior to the release of software to configuration control, SQA organization must certify the software's traceability of requirements from the DPSR through the code/build release phase. By this process, SQA ensures the incorporation of all approved changes into the software and documentation. Additionally, once the software media is released to configuration management control, the SQA maintains one or more copies of the central vault working copy in a separate location for added configuration control. Lastly, SQA has the responsibility of auditing the CSA organizations records as part of the periodic Configuration Management (CM) audit to ensure compliance of procedures with specified policies.

This chapter has discussed the Patriot software development management method. In the next chapter, the Patriot software development method will be used to illustrate how the Cosmos model can be implemented for military software development management. The idea is to determine areas of correlation between the Patriot method and the model, and to determine what tools used by the Patriot method might benefit future military software development management programs.

IV. COSMOS IN DOD ANALYSIS

A. CONTEXT OF THE ANALYSIS

The objective of this chapter is to illustrate how Cosmos model can be used in a DoD environment by providing examples of how the Patriot software development management method accomplishes concepts presented in the model. Additionally in this chapter, this thesis will determine types of tools used by Patriot software development management method that can be used for the management of future military software development projects. Considering that the Cosmos model represents a model presenting a "holistic" view of the software development process, the events within Patriot software development method will be related with the six principles of dealing with dynamic complexity set forth in this model.

To effectively organize the relationship of the six principles with the Patriot software development method, the analysis will be divided into the three distinct dimensions of the Cosmos model: Activity, Communication, and Infrastructure. These dimensions will provide the framework with which to visualize the applicable principles.

Additionally, this analysis discusses how the Patriot software development method demonstrates another key aspect of the Cosmos model. Specifically, the Patriot software development management method will be observed to determine if it demonstrates the interaction of the Cosmos model's three dimensions.

The analysis begins with the discussion of the six principles within their related dimensional context. This is followed by treatment of the other key aspect of the model with respect to the Patriot software development method. The analysis will be based on information presented in Chapter II which defines and discusses the Cosmos model, as well as the information presented in Chapter III concerning the Patriot software development management method.

B. ACTIVITY DIMENSION

As stated in chapter II, this dimension allows insight into the trade-off of Flexibility versus Stability. To accomplish this, the software development management process must take into consideration three of the six principles for managing dynamic complexity, namely: separation of concerns, coevolution, and protoiteration. The activities demonstrating Patriot's treatment of these three principles are categorized with respect to control level and execution level activities.

1. Separation of Concerns

The principle of separation of concerns deals with the concept of breaking a problem or project up into subproblems to "divide and conquer" a project, or, in other words, to effectively manage a project's complexity.

Due to the generally large and complex nature of current and expected future military software development projects, there is a distinct need within DoD to establish methods that will allow such projects to broken into manageable parts. At the control level, DoD projects need ways to accomplish software development, configuration management, and quality control. At the execution level, a systematic approach for actual software development is needed to ensure that system requirements are methodically translated into accurate software code.

To satisfy such needs, within the Patriot software development management method, activities at both the execution level and control level have been established to accomplish the principle of separation of concerns.

At the execution level, the Patriot software development management method uses several activities which demonstrate the separation of concerns principle. Most obvious is the use of MIL-STD-2167A as a basis for the actual activity of software development.

MIL-STD-2167A, through its call for specific documents in specific formats, reflects the idea of a waterfall

approach to software development. Software development is decomposed into the activities of requirements analysis and determination, software design, to include preliminary and detailed design, coding, integration testing, and formal testing. This method of software development decomposes the software project into subproblems that can be more easily dealt with. The waterfall model approach divides up the overall software development activity into a sequence of sub-events that follow in a logical progression.

Furthermore, the identification of software is divided into Computer Software Units (CSU), Computer Software Components (CSC), and Computer Software Configuration Items (CSCI) components. Breaking up the software design effort into CSUs, CSCs, and CSCIs allows the programmer to dissect the overall program into chunks of functionality that are easier to work with. While easier to code, the CSUs must always be developed with the understanding that they will have to be integrated into CSCs and ultimately CSCIs at a later time.

Also at the execution level, the principle of the separation of concerns is demonstrated in the Patriot software development method by the way the transition to Ada is being handled. To transition to Ada, members of the Patriot Program Office (PPO) have taken an incremental approach. They have started the transition by first attempting the conversion on a small yet central processor intensive portion of the Patriot software. Using this smaller portion as a test case, the project is able to reduce overall complexity by limiting the number of lines of code that have to be converted, enabling lessons learned from this case to be used in future code transition.

On a more macro-level, the approach of making the transition to Ada a separate project as opposed to attempting to integrate the transition into the development process, also demonstrates the consideration of separation of concerns. This separate project allows the transition to occur on a timeline that is not necessarily tied to the

software development timeline established for system software development, again reducing the complexity of the process.

Additionally, the use of the Ada programming language allows software developers to define software by code objects. These objects allow the coding task to broken down into code structures that are easier to work with and modify.

The control level of the separation of concerns principle is demonstrated in the Patriot software development management method by the project's chosen organizational structure. The division of the organization into subunits by functional area allows each division or section to focus on and become expert in one area. For example, the Configuration Management, Product Assurance, and Software Laboratory Organizations, and their subdivisions, all have separate and identifiable missions. Focusing on only a certain portion of the overall project is less complex than having to deal with all aspects of the software development method at one time.

2. Coevolution

The focus of the principle of coevolution is the concept that activities must develop in conjunction with one another. This idea is broadened by the realization that all requirements cannot be known up front, and that they must evolve as the architecture and the design of the project evolve.

Within DoD, due to the number of stakeholders of a software project as well as the evolving nature of the threat, initial software requirements are generally fuzzy and do not consider all possible needs. Because of this, a software project's requirements can be expected to evolve and change several times during its lifecycle. To accommodate needed software evolution, DoD software development methods must establish ways for allowing input into the development process to occur. Additionally, at the execution level, DoD software development activities must have methods that ensure that subproblems are developed with a common thread so they

can be reintegrated at a later date.

The Patriot software development management method has specific ways in which this is accomplished. Demonstrating this principle, at the control level, the Patriot software development management method utilizes the Software Problem Report (SPR), Software Investigation Report (SIR), Software Assessment Report (SAR), and the Deviation Waiver (DW) processes. These processes allow inputs to be integrated into the software development process from the software user and the various organizations which review and have responsibility for accurate software development. By having these processes, the software development management method deals with the reality that requirements can change or evolve as information about the threat, more capacity in hardware, and improvements in technology become available.

Looking at the engineering services contract, the concept of coevolution can also be seen. The contract allows for quick reaction to the need for upgrades to the system software. By having a number of prepaid man months readily available, the Patriot software development management method recognizes the idea that changes will occur and that an efficient method for dealing with them is needed.

At the execution level, a relationship between the principles of coevolution and separation of concerns can be seen in the Patriot software development management method. During integration testing, the idea of the CSUs being brought together to form a CSCI demonstrates the idea of coevolution. As elaborated in the Cosmos model, all CSUs must be developed in conjunction with one another with the same common goal if they are to be effectively integrated for successful software development.

3. Protoiteration

The principle of protoiteration deals with the concept that the right solutions to dynamically complex problems are rarely found on the first try. Specifically, this principle expresses the use of prototyping in an iterative fashion as a

way of better defining solutions to problems. This concept also takes into consideration the idea that subproblems must be properly structured so that they can more easily accept an iterative change.

Because of the fuzzy nature of initial software requirements for large military software development projects, the use of prototyping in an iterative fashion within DoD to assist in clarifying these requirements is very beneficial. Also, because of the changing nature of military software requirements, software should be constructed to ensure changes to the software can easily be made.

A direct demonstration of protoiteration is seen in the Patriot software development management method's use of prototyping during software requirements definition. In an attempt to realize all the possible aspects of the software being developed, the Software Laboratory utilizes prototype software to allow engineers to better visualize what the software is capable of accomplishing and how it might be accomplished.

A complement to this which has already been mentioned is the Ada programming language. Ada's way of defining code in modular packets allows for changes to be made to specific packets, thereby making changes to the overall software much easier. This programming language, currently being used to develop Patriot software, creates subproblems or CSUs that are structured to more easily accept changes compared to a language that does not have a modular construct.

The idea of the principle of protoiteration is also seen by the way the Patriot software development management method defines software. By defining software in terms of developing CSUs, CSCs, and CSCIs, the Patriot software development management method takes into consideration structuring subprojects which more easily accept change. By defining software development in terms of specific functions and creating a specific CSU for that function, the software development management method allows developers to look at smaller areas of code and software documentation when

attempting to institute a change. This follows the idea of structuring subproblems to more easily deal with change.

The use of Software Development Files (SDF) by the Patriot software development management method also provides a demonstration of the principle of protoiteration. SDFs archive all relevant information concerning a CSU. By doing this, they take into account risk management information and source code structure that can be looked at, possibly easing the development of software in future iterations.

Facilitating the principle of protoiteration in the Patriot software development management method, are the structures of the SPR, SIR, and DW processes which accomplish coevolution. These processes take into consideration that requirements inputs, and therefore changes to the software can happen at anytime during the software development cycle. By allowing for changes to be input into the development method, these processes accommodate the iterative concept of protoiteration.

By having methods and structures that accomplish the three principles mentioned above: separation of concerns, coevolution, and protoiteration; the Cosmos model states that a software development method is able to have several concurrent software development activities ongoing. Specifically, one software development activity could be in the requirements stage, while another is in the design stage, while yet another is in the coding/testing phase of development.

Evidence of this in the Patriot software development method can be seen in that usually three Post Deployment Builds (PDBs) are ongoing at any one time. Currently, PBD-5 is in the requirements phase of development. At the same time, the Patriot software development organization is in the process of integration testing on PDB-4, while it maintains and provides fixes to software, through the SPR, SIR,and DW processes, that are needed in the fielded PDB-3 software. [Ref. 32]

Although having methods and structures that accomplish

the concepts of the activity dimension provides a software development management method with effective ways to accomplish software development, these methods and structures cannot effectively be used by managers without also considering the communications dimension. The communication dimension provides for the intricate coordination among responsible software development organizations that is required to successfully coordinate and manage concurrent activities and the actions presented in the examples above.

C. COMMUNICATION DIMENSION

As stated previously, the communication dimension allows for insight to be gained into the Modularity versus Interconnectivity trade-off. To accomplish this, two principles of controlling dynamic complexity must be considered: inclusion and reification.

By creating tools (e.g. role maps) that accomplish the concepts of these two principles, the communication dimension affects the interaction of all three dimensions. It also reinforces or limits their development depending on how effective the communication dimension is treated. Specifically, a model of a software development method's communication structure makes communication an explicit versus an implicit activity.

1. Inclusion

The principle of inclusion deals with the concept that all individuals and organizations that are stakeholders in the software being developed must be considered as to their needs and responsibilities. The rationale for this is that participation by all concerned is seen as beneficial to problem identification.

Because of the number of organizations both in and outside the military that have a stake in the development of most large software projects, the customer driving the efforts of the project is, many times, not well defined for the software development project manager. Because of this,

the need to ensure their inputs can be considered is imperative. For example, a stakeholder outside the military is Congress. To manage effectively, a software development project manager must understand the stake that congressional representatives have in his or her program. This will assist in allowing the manager to determine exactly what requirements they might place on him or her before appropriated funds for the project are released. Within DoD, the software project manager must not only consider the user, but also organizations like the Independent Test and Evaluation and Independent Verification and Validation (IV&V) organizations. Approval by both of these organizations is required before the software product can be released to the user. To allow inputs to be considered from these organizations, both in and out of the military, the software development process must establish methods that allow them to be captured.

Within the Patriot software development management method, consideration for the principle of inclusion is seen in the use of a formal review process and the use of the SPR, SIR, and DW processes. The formal review process of the Patriot software development method allows the organizations with software development responsibility to provide input to the process of software development. For example, the formal review process provides a channel for the Software Quality Assurance (SQA) organization to inspect the accuracy of the software as it relates to the requirements to ensure the software is ready to continue into the next phase of development. The In-Progress Review (IPR) provides the user and several other stakeholders, including the IV&V contractor, a method of ensuring that his or her comments and needs (e.g. cost, schedule, and performance) are clearly understood. Additionally, the SPR, SIR, and DW provide channels in which organizations outside the Application Software Department (ASD) can provide feedback to the development team. For example, the Software Configuration Management organization (SCM) utilizes SPRs in addition to

Engineering Change Orders (ECO) and Error Reports (ER) to describe problems found with software media to ASD and to ensure that control is maintained over the incorporation of the fixes.

The Patriot software development method also shows the principle of inclusion by the early incorporation of the IV&V contractor into the software development process. The close working relationship between the independent inspection organization and the software creator provides an extra set of unbiased eyes to ensure that the development method is providing what the user wants in the best possible fashion.

2. Reification

The principle of reification deals with the concept of identifying the objectives of an activity in a clear manner by stating reasons why a particular activity must be accomplished. This allows persons responsible for the accomplishment of the objective to do so from a perspective of truly understanding the ramifications of the objective.

Within DoD, due to the public nature of the organization, there are many individual and organizational stakeholders with varied objectives involved with large software development projects. Due to these numerous organizations involved in the acquisition process with oversight of a software project, the objectives of a software development project designed to satisfy user's needs have a tendency to become blurred. Because of this, there is a distinct need to develop methods to ensure that all relevant personnel receive required information to ensure that a project's user objectives are reified.

Also, because of the short duration (two to four years) of military assignments within DoD programs, institutional knowledge of objectives and communication structures have the potential becoming lost. Because of this, DoD needs to establish ways to ensure such concepts are captured to ensure the smooth transition of software project management from one regime to the next.

To accomplish this, the Cosmos model calls for the use of modeling roles within the communication structure. The "role map" deals with identifying who must communicate with whom, why they must communicate, what they must communicate, and how they must communicate to ensure that the required information is disseminated properly to allow for it to be thoroughly understood. This will translate into effective software development. Role maps are described by the Cosmos model authors as actual wire diagrams depicting these aspects of communication so that the communication structure of a project will be *explicitly* defined.

During the investigation of the Patriot software development management method, the author of this thesis did not find specific role maps as defined by the Cosmos authors for the identifying communication structure within Patriot software development management activity. What was found, however, was an *implicit* communication structure in the Patriot software development management method.

Through the use of formal review process, the SPR, SIR, DW, SDF, ECO, ER, and the Data Item Descriptions (DIDs), an understanding of the implicit communication structure of the Patriot software development management method can be obtained. The formal review process establishes communication between the varied organizations with software development responsibility as mentioned earlier. Within the Patriot Software Development Plan (SDP) there exists suggestions on how to structure and which topics to include in the different reviews. Also covered are suggestions for the minimum participation of key individuals to attend several of the reviews. Further, the SPR, SIR, DW, SDF, ECO, ER, and DIDs all identify what specific information needs to be passed on to the using organizations and in what format. Although not specifically a role map, as described in Cosmos, much information is gained about the communications structure of the Patriot software development management organization by understanding the use of formal reviews and reports, waivers, orders, and descriptions mentioned above.

The problem seen with allowing the communication structure to be implicitly defined is the difficulty of readily seeing the effects of lack of communication in a specific area and the effect on the rest of the communication structure should it occur. To avoid this, one would need enough familiarity with the overall software development process to enable him or her to visualize the effects. Through the use of the role map of the communication structure, however, an observer could readily understand the effects that a lack of communication would cause by observing the nodes on the wire diagram that would be neglected should communication channels break down.

Although the activity and communication dimensions of the Cosmos model provide some understanding of the concepts required for the management of software development, they do not cover the very important aspect of infrastructure.

D. INFRASTRUCTURE DIMENSION

The infrastructure dimension deals with the concept of providing frameworks for taking into account what processes are needed to achieve a software project's objectives. This dimension provides insight into the trade-off of a Broad- versus Narrow-Scope focus on issues. To effectively make this trade-off, tools that capture the concept of the principle of continual improvement must be created.

1. Continual Improvement

The principle of continual improvement deals with the idea that all things can be improved upon. To accomplish this, frameworks or infrastructure that make this possible must be established.

Currently, most software development within DoD is contracted out to civilian development organizations. Considering that most of these organizations are at a Software Engineering Institute (SEI) process maturity model level of II or less, [Ref. 20:p. 277] there exists a distinct need within DoD to establish methods that capture

process information for software development. Also, because of the current lack of trained military personnel in software management and the limited institutional knowledge of DoD civilians in the area of the software development process, DoD is not in a position to assist contractors with the continual improvement of software development. This is especially true now considering that the use of military standards, with their mandated deliverables, is no longer required. Due to this, it is imperative that military software development projects establish infrastructure that captures vital process information to be used for continual process improvement.

The Patriot software development management method has several structures that accomplish continual improvement. Within the Patriot software development management method, the organizational design of the Missile System Division's organizations that are responsible for software development demonstrates infrastructure that takes into consideration the principle of continual improvement. This can be seen in the way the functions of the organizations are divided to provide checks and balances to ensure the established standards are met, and that identified requirements flow through the development process and are captured in the functions of the developed software programs. For example, the responsibility of the SCM is to ensure that the changes made to the software media by the ASD are properly controlled and incorporated only after all responsible parties agree to the changes, and the proper archives have been updated.

A more direct demonstration of the principle of continual improvement being accomplished through infrastructure is the Patriot software development management method's use of Software Initiative Working Groups (SIWGs). These working groups have been established for the expressed purpose of observing the processes of risk management, project management, subcontractor management, defect analysis, requirements management, peer reviews, trusted software, process measurement, process definition, technology

management, and software quality assurance. Much of the information concerning the above processes is gathered through the use of STEP Metrics. Once observed, SIWGs then find ways in which to make these processes more effective. The end result of the SIWG's actions is the improvement of the overall software development process.

Another infrastructure of the Patriot method that demonstrates continual improvement is the process of risk management. The Patriot software development risk management process utilizes a taxonomy of risks and risk mitigation methods for software development. These taxonomies not only provide information on types of risks and ways that they have been dealt with in the past, but also allow for the incorporation of new unpublished risks and mitigation techniques. This improves the ability of the software development management method to accomplish risk management which in turn improves the overall software development process.

Similarly, the infrastructure established to accomplish software reuse considers the principle of continual improvement. By reusing known, valid software code, the Patriot software development management method decreases risk associated with developing associated CSUs. Hence, reuse has the potential to improve the efficiency of software development.

Although personnel training might be considered an intangible infrastructure for software development, it does illustrate continual improvement. Better trained personnel are better prepared to handle complexity and change because they have a broader base from which to draw on for solutions to problems. This in the end can have a positive effect on the continual improvement of developed software. Reflecting this, the Patriot software development management method has established training requirements and methods for identifying and providing for additional training needed by its personnel to accomplish the various aspects of software development and management as mentioned in Chapter III.

E. INTERACTION OF THE THREE DIMENSIONS

A key aspect of the Cosmos model is that the three dimensions are all interrelated as the principles of dealing with dynamic complexity found within these dimensions are considered. This concept can be seen in the Patriot software development management method through the examples below.

While treating the division of labor among the organizations and considering the principle of separation of concerns within the activity dimension, the idea of having different organizations work various functions of software development demonstrates the aspect of infrastructure. Related to the activity and infrastructure dimensions is a communication structure among the various organizations required to allow these organizations to accomplish their software development tasks. This is accomplished by having tools which accomplish the principles of inclusion and reification found in the communication dimension.

Another example of how the Patriot software development management method demonstrates the relationship among the three dimensions is seen by observing how it accomplishes the principle of coevolution within the activity dimension. Specifically, the SPR and the SIR processes demonstrate that the Patriot software development management method realizes that not all requirements can be known up front. These processes are also part of the infrastructure that provides for continued improvement. They also help to implicitly define the communication structure of the software development management method.

Through the numerous examples found in the Patriot software development management method that demonstrate consideration of the principles in the activity and infrastructure dimensions, the Patriot software development management method closely relates with the Cosmos model. In the communication dimension, however, this same correlation is not found. Although the concept of a communication structure is dealt with in the Patriot software development

management method, it is treated only implicitly. This is contrary to the explicit role mapping of the communication structure the Cosmos model's authors call for.

However, because of the close correlation of the activity and infrastructure dimensions considering the implied treatment of the communication structure, the Patriot software development management method demonstrates a relatively holistic method of software development management with respect to the holistic method described by the Cosmos model. Evidence of the capability of the Patriot software process and its relatively holistic nature is seen in the ability of the development management method to effectively manage the complexity brought on by the changes to requirements during the Gulf War. During the Gulf War, four system software changes occurred in a short five month period. Credit for these quick changes was given to what was called software responsiveness. During a Gulf War After Action Review (AAR), statements concerning how the software was developed for responsiveness described several of the key concepts which are addressed in the Cosmos model. [Ref. 44]

One specific example concerned bringing the IV&V contractor on board early in the project development cycle. This idea begins to demonstrate the idea behind principle of inclusion. In order to effectively manage the software development process, the manager must have the input of all stakeholders to the project. By including the IV&V contractor early on, the developer has an independent set of eyes offering suggestions about the software development. Also, by including the IV&V contractor early, the developer is making a partner out of a potential adversary. Since the software must be "blessed" by the IV&V before it can be released, the early inclusion of the IV&V can speed up the development process.

Also noted in the AAR, was the importance of experienced, competent software personnel. The need for well trained persons with system specific knowledge and ability to effectively employ the programming language, tools, and

methodologies demonstrates the principle of continual improvement. By having a training infrastructure, people can be continually educated to provide the needed talent to accomplish quick software updates in a continually changing environment.

Further, the principle of continual improvement through infrastructure was again touched on by briefers when they noted the need for decision makers to have a willingness to take calculated risks with respect to software development. During the Gulf War, software development managers took a calculated risk and sent upgraded software to the field prior to completing the desired level of testing. The rationale for this was that the testing accomplished had shown the software to be capable and that further indepth testing, as is usually required, would cause fielding delays that might result in the loss of lives or defended assets due to missed SCUD engagements.

In summary, the ability of the Patriot software development management method to quickly respond in the fluid environment of the Gulf War demonstrates a successful software development method that offers specific actions and processes that can be used for future large military software development projects.

F. TOOLS FOR SUCCESSFUL DOD SOFTWARE DEVELOPMENT MANAGEMENT

This thesis has shown that the Cosmos model for software development management is a model that provides a holistic view of the software development process. Additionally, this thesis demonstrates how the Patriot software development management method relates to the Cosmos model. Because of Patriot's close relationship with the Cosmos model, the activities and processes that are utilized by this military software development management method provide a relatively holistic set of software development management tools. These software development management tools represent the types of tools that, when used by future military software development

projects, should offer similar software project success. The rationale for this is that these tools, to include the role map, provide the software development manager with ways of dealing with the three essential trade-offs of the Cosmos model: Flexibility versus Stability, Modularity versus Interconnectivity, and Broad- versus Narrow-Scope.

The paragraphs that follow provide an explanation of each of the tools that could be utilized by future large military software development projects.

1. Engineering Services Contract

The Engineering Services Contract provides a vehicle which allows a software development project to quickly implement changes to the software baseline. By purchasing a block of man-months that can be used as needed in the future, the project can quickly get to the act of software upgrade and avoid the contractual "red tape" that can slow the process down.

From research into the Patriot software development project, the Engineering Services Contract is found to be a key factor in enabling software development to occur as quickly as it does. Without this contract, it is estimated that the software development process would take an additional eighteen months over the current eighteen month time frame for a normal software development cycle. [Ref. 33].

Therefore, the Engineering Services Contract allows a manager to deal with the issue of flexibility/stability. The use of this contract allows a manager to gain flexibility in the planning process. While the nature of a large software development project requires that rigid planning occur early on in order to gain stability and understanding of what must be accomplished, these plans are usually unable to consider all possible contingencies. The Engineering Services Contract, by having man-months readily available, allows the manager to react quickly to unforeseen events that occur during the software development cycle. This offers the

manager of a large software project the flexibility to accommodate change while allowing plans to be created early on in order to stabilize the direction of the project.

2. Post Deployment Build (PDB) Software Development Method

The PDB method allows a software development project to accept input from the various stakeholders, and it provides a tool to optimally develop and test software code. The PDB method is made up of many subprocesses, structures, and activities that interact to create accurate system software.

While the Waterfall model of software development provides a basic logical sequencing of the software development activity, the SPR, SIR, and DW provide additional important processes that allow for needed corrective input to occur during the software development cycle. These reports and waiver are available for documenting the need for a corrective action depending on at what point during software development activity a stakeholder realizes that there is the need for a corrective action to occur.

For example, the DW documents the fact that during integration testing an octal patch to the object code of the software program had to be made to allow the program to run as required. This waiver would then be used to begin the process of having the source code changed to match the object code patch during a subsequent PDB cycle.

If during integration testing, testers realize that there is a need for a non "show-stopper" corrective change to the software program, a SPR is issued. The SPR is then used to begin the process of implementing the correction into the source code in a subsequent PDB. SPRs are also used to document any problems and begin the correction process when problems with the software are realized during requirements, design, and coding phases.

SIRs normally document requests for changes to the software that are voiced by stakeholders outside the Application Software Department. For example, when a user

expresses the need for a change to the DPSR a SIR is issued to document the request change. The SIR is then used to begin the correction process.

The correction process begins with the convening of a Software Review Board (for SPRs), or a SIR Review Board to determine whether the requested change or corrective action is actually needed, can be accomplished, and to decide when an accepted change should be implemented. If it is decided that the change is required the software development activity of the PDB process is initiated and required software is developed.

The PDB process also consists of the use of CSUs, CSCs, and CSCIs. The use of CSUs, CSCs, and CSCIs as defined by MIL-STD-2167A, although appearing axiomatic with respect to coding any software program, provides the basis for a "Call" which begins the important process of integration testing. This process of integration testing at each level (CSC, CSCI) of functionality ensures that almost every problem with the software code is found prior to Informal and Formal Qualification Testing. Since Formal Qualification Testing is accomplished by an independent test agency that reports to a chain of command outside the stakeholders included in the development process, the need for software problem resolution prior to this is imperative. The potential outcomes of Formal Qualification Tests riddled with software problems could be the delay of future funding or even the discontinuation of the software development project. [Ref. 45]

The review of the integration testing data provides information that is incorporated into the SDF. The SDF which is an artifact of PDB process provides a location where specific information concerning CSUs is stored. In addition to integration testing information in applicable SPR, SIR, and DW format, the SDF holds CSU source code and risk management information. While providing for configuration management, the SDF is a vital source of information for software reuse by providing the type of information on which

the decision for software reuse is made.

The use of the programming language Ada is also a vital part of the PDB process. In addition to being required by law unless a waiver is granted, the use of Ada provides for modular programming that adds to the ability of the correction process to quickly implement changes to a software build. Although the use of Ada has not yet been fully implemented in the Patriot project, the risks overcome in doing so are expected to be outweighed by the benefits associated with the positive attributes of the language. An example of this is the previously cited ease of software reuse.

The use of the PDB process while offering a manager methods for dealing with the Flexibility versus Stability trade-off through the use of methods that allow for separation of concerns, coevolution, and protoiteration, also offers insight into the modularity/interconnectivity issue. The PDB process allows the software development process to be broken into subdivisions while providing communication methods which allow interconnectivity to occur. Through the use of a tailored MIL-STD-2167A type construct requiring requirements definition, software design, coding, and test, the process is broken into logical modular subproblems. The process offers methods for providing communication among the various subissues through the use of the review processes and the SPR, SIR, and DW processes.

3. STEP Metrics

To monitor the activity of the software development method and to manage risk in the areas of cost, schedule, and performance, managers of the Patriot software development method utilize the metrics contained in DA PAM 73-1 which is the Army's manual defining a set of metrics called the Software Test and Evaluation Panel (STEP) metrics. [Ref. 32,43]

STEP metrics provide a tool that is used to determine if the software has achieved the required level of functionality

and maturity to proceed to the next stage of development or test. This occurs because STEP metrics provide for the continuous measurement of the process throughout all phases of the software life cycle. These twelve metrics provide both process and product measures through consistent interpretation and description of software status in formats that are objective, timely, and finite. [Ref. 43]

Currently, STEP metrics are briefed during monthly Program Reviews and during the IPR held to update the user on software development progress. This schedule and the provided metric data is considered timely enough and in the proper format to ensure effective management of software development events. [Ref. 32]

The use of STEP metrics provide the manager with a tool to assist in deciding the broad-/narrow-scope issue. These metrics provide a manager the means to realize the strengths and weaknesses of the software development process. This information can be used to determine what immediate changes must be made to the process as well as to determine strategies to improve inefficient procedures in the future as capability to do so becomes available.

4. Risk Management Taxonomy

While risk management is required by DoD Directives for all software development programs, the use of a taxonomy of risks and risk mitigation solutions is an effective method for maintaining the artifacts of Patriot software development risk management process. It provides a starting place from which the risks associated with a particular type of software can be known prior to the software being developed.

Also important in risk management is the use of the SDF to capture the risk assessment data. This data in the SDF is used in the decision of whether to possibly reuse a piece of software in a future software build.

The Risk Management Taxonomy and SDFs assist the manager in making the Broad versus Narrow-Scope trade-off. These infrastructures provide methods for institutionalizing

information concerning specific actions that where taken to accomplish or improve software. This ensures that persons responsible for software development do not repeat costly mistakes made in the past. These infrastructures give the software development management method a continually growing base from which to improve.

5. Software Development Library

The Software Development Library is an electronic data base depository for the storage and controlled access to all code, test specifications, test results, and project SDFs. This library provides a location for the type of information that is considered to be reusable. Like the Risk Taxonomy, considering that software reuse has the potential to save time in software development process, the value of such a tool and its ability to assist the manager in making the Broad- versus Narrow-Scope trade-off can readily be seen.

6. Project Organizations

The organizations involved in the development of Patriot software accomplish functions that provide the manager with independent yet interrelated views of the software development activity. The SWL, SCM, and SQA provide a set of checks and balances that ensures that the developed software is as error free as possible before being released to the field.

The SWL is responsible for software design to include related documentation, coding, unit testing, software development process improvement as well as software integrating and validating. Additionally, SWL analyzes and assesses all the software requirements and creates some requirements for specific CSCIs. To accomplish these functions the SWL is further subdivided into four separate organizations as outlined in Chapter III.

The SCM organization controls software configuration through configuration identification, change control, interface compatibility, and status accounting as mentioned

previously. Although an independent organization, the SCM works closely with the SQA organization to ensure the methods used to accomplish configuration management conform to the policies and specifications outlined in related corporate directives.

The SQA organization is responsible for software product evaluations. These product evaluations ensure that the software meets the requirements set forth by the Software Development Plan and other internal policy documents as well as those of the contract with respect to content and format. The SQA is independent of specific development organizations they support. Therefore, although the SQA works closely with the software engineering effort, it is managed and reports through an independent chain.

While not specifically an internal software development organization, the IV&V contractor is part of the military software development process. Bringing this organization on board early in the software development cycle provides for additional input from an unbiased organization that can assist in spotting deficiencies that could be costly later on.

These organizations allow the manager to make both the Flexibility versus Stability and the Broad-versus Narrow-Scope trade-offs. The flexibility/stability issue is dealt with since the organizations are independent and flexible, allowing them accomplish their tasks in the most optimum manner while also providing unbiased consistent views of the different aspects of the software development process. The broad/narrow-scope issue is dealt with by SIWGs within the SWL organization which provide process information on various areas of the software development method. This information is used to continually monitor and improve a software development method's performance, and it allows a manager to understand where the process is now and gives a clearer picture of how to get to a desired level of performance in the future.

7. Personnel Training Program

Having experienced, competent people was seen as a reason for the success of Patriot software development management method during the Gulf War. Each organization that is part of the software development process has established training programs that take qualified personnel and ensure that they receive any additional requisite training either prior to or during the development of specific software. This gives the manager insight into making the Broad versus Narrow Scope trade-off. By providing a method to ensure that personnel are continually gaining understanding of new and better technology to be used in improving software development, a training program forces the manager to consider training as a factor that must be figured in when creating timelines for software development. Hence, a manager is forced to take a longer view of the software development process.

8. Role Maps

Although not explicitly found in the Patriot software development management method, a role map of a project's communication structure as described in the Cosmos model provides explicit insight into who, what, when, why, and how communication should occur during software development. The role map provides for the quick diagnosis of problems or potential problems should a communication node discontinue functioning. This tool of a software development method is key in identifying where coordination among independent stakeholders in the development of software is needed.

Role mapping offers a tool for the manager to use in accomplishing the Modularity versus Interconnectivity trade-off. Through the use of a role map, a manager can realize what communications links must be created to allow separate project modules to pass vital information among them.

The following chapter, Chapter V, will consider recommendations that can be drawn from this analysis as well as present areas for further study.

V. RECOMMENDATIONS AND AREAS FOR STUDY

A. SUMMARY

This thesis has examined how the Cosmos model represents a holistic view of the software development management process, and how it can be used as a basis for future military software development management. This was accomplished by analyzing the current state of the software development process in both the military and civilian sectors, and finding that the emphasis is currently being placed on only one facet of the overall process. From this, this thesis identified a need for a more holistic approach that encompasses both the production and management facets of the software development process, using the Cosmos model as a possible solution. A description of the Cosmos model was presented, providing a comprehensive view of the software development process through its use of three dimensions and six principles which allow a manager to make three essential trade-offs. This was followed by a description of the Patriot software development management method. This description was used to provide examples of how the Cosmos model concepts can be used for DoD software development management. Based on the analysis of the Patriot examples, this thesis recommends eight significant types of tools that could be used by future military software development projects to ensure that a holistic approach to the software development process is taken. Within Patriot, these tools are: an Engineering Services Contract, a Post Deployment Build (PDB) Software Development Method, a Risk Management Taxonomy, Software Development Library, Personnel Training Program, Project Organizations, STEP Metrics, and Role Maps.

B. RECOMMENDATIONS

1. DoD Policy Recommendations

Based on the information contained within MIL-STD-498 and the findings summarized above, it appears that the

Secretary of Defense's recent order to discontinue use of military standards for military product development may have been a set-back in the area of software development management. MIL-STD-498 was created as a replacement of MIL-STD-2167A and other military standards governing various aspects of the software development process. This was done in attempt to resolve the problem of these previous military standards not clearly emphasizing both sides of the software development process. MIL-STD-498 accomplishes this though its use of specific requirements which force a manager to address both the production and management facets of software development process. [Ref. 27:p. i]

MIL-STD-498 is intended to be an all inclusive standard in the area of DoD software development management requiring a holistic view of the software development process. Essentially, this standard improves compatibility with non-hierarchical design methods; improves compatibility with computer aided software engineering (CASE) tools; gives alternatives to, and more flexibility in, preparing documents; provides clearer requirements for incorporating reusable software; enhances supportability; and improves links to systems engineering. This standard does not specify or discourage the use of any particular software development method. This leaves the developer with the responsibility for selecting software development methods that support the achievement of contract requirements. Importantly, this standard is meant to be tailored by the program office or other DoD agency to ensure that only necessary and cost-effective requirements are imposed on software development efforts. [Ref. 27:p. i]

While the standard attempts not to limit the program office to any specific software development management model, it does require that the contractor create a software development management process that includes specific activities. These activities, however, may overlap, may be applied iteratively, may be applied differently to different elements of software, and need not be performed in the order

listed below: [Ref. 27:p. 12]

1. Project planning and oversight
2. Establishing a software development environment
3. System requirements analysis
4. System design
5. Software requirements analysis
6. Software design
7. Software implementation and unit testing
8. Unit integration and testing
9. CSCI qualification testing
10. CSCI/Hardware Configuration Item integration and testing
11. System qualification testing
12. Preparing software for use
13. Preparing for software transition
14. Integral processes: Software configuration management, Software product evaluation, Software quality assurance, Corrective action, Joint technical and management reviews, Other activities (e.g. Risk management, Use of metrics, Personnel education, and Reuse)

Although specific requirements in each of these activities must be accomplished by the developer, the emphasis lies on the development and recording of planning and engineering information, an intrinsic part of the software development process, to be performed regardless of whether a deliverable is required. Further, the idea is to tell the developer the "what" of the requirement but not the "how" of getting the requirement accomplished. [Ref. 27:p. 12] For example, the standard says that the developer must perform risk management throughout the software development process. Additionally, it states that:

> The developer shall identify, analyze, and prioritize the areas of the software development project that involve potential technical, cost, or schedule risks; develop strategies for managing those risks; record the risks and strategies in the software development plan; and implement the

> strategies in accordance with the plan. The developer shall identify and define a set of software management indicators, including the data to be collected, the methods to be used to interpret and apply the data, and the planned reporting mechanism... [Ref. 27:p. 26].

While giving specific guidance that risk management must be accomplished the standard does not state that the developer must use, for instance, STEP Metrics to accomplish this task. Although not specifically requiring this set of metrics, the standard does provide it as an example emphasizing that it is only an example and not required to be used. Examples are also provided for other areas, such as Joint Management Reviews.

The standard is written to accommodate all sizes of software development projects to include large projects that have several different builds. It is also written to accommodate projects with different acquisition strategies such as Grand Design, Incremental, and Evolutionary. The standard provides examples of how it can be used with these strategies and gives guidelines for the scheduling of selected activities (from the 14 mentioned above) in each build.

The standard ends its discussion with a warning to managers about limiting their software development and management flexibility. For example, one common mistake that is made is to treat all CSCIs as though they must be developed in "lock-step", all designed by a certain date, implemented by a certain date, etc. This can result in a development process that limits optimum software development. Flexibility in scheduling gained by decoupling CSCIs from the same schedule can be effective in avoiding this mistake. The standard reiterates that the activities in each build should be laid out in a manner that best suits the work to be done. [Ref. 27:p. 40] Also, care must be taken to ensure that the flexibility inherent in the use of the standard is not nullified by rigid scheduling of the Contract Data Requirements List (CDRL). If the CDRL lays out a strict

"waterfall" sequence of deliverables, little room is left to propose an innovative development process, and CSCIs are forced into a lock-step and potentially suboptimum order of development. [Ref. 27:p. 56]

Considering this information, MIL-STD-498 presents concepts and examples of activities that force a manager to manage with a complete view of the software development process. By no longer requiring this standard, the SECDEF may be contributing to the continued ineffective state of software development management. The author of this thesis recommends that the SECDEF provide for an exception to this policy with respect to this military standard and require its use in the development of military software products.

This author further recommends that, at the very least, the eight types of tools described in Chapter IV of this thesis be incorporated into the future editions of DoD Directive 5000.1 or DoD Instruction 5000.2. This should be done in such a way as to require the concepts of each tool to be addressed by the unique methods used for software development described by a project's software development plan.

2. Recommendations for Patriot

First, while the lack of role mapping to provide for an explicit communication structure appears not to severely hinder Patriot software development management, the explicit treatment of the communication dimension, as defined by the Cosmos model, would offer additional insight into the management of software development. As stated previously, role mapping would provide an accurate and efficient method for a manager to quickly realize communication deficiencies or redundancy. By acting on these issues, the manager could streamline the software development process which could in turn allow for more efficient creation of software programs. This statement considers that the Patriot software development management method already demonstrates thorough treatment of the activity and infrastructure dimensions.

A complement to the idea of explicit treatment of the communication dimension is the concept of the Patriot Project Office (PPO) realizing that its actions are captured by the Cosmos model. By realizing the Patriot software development management method's relationship to the Cosmos model, the program manager can gain a conceptual holistic understanding of how the various aspects of the software development management method interrelate. As shown in Chapter II, this comprehensive understanding provided by the Cosmos model appears to be a complete way to realize the management requirements for a large successful software development project.

C. AREAS RECOMMENDED FOR FURTHER STUDY

1. Creating a prescriptive model for large software development projects.

In looking at the Patriot software development management process, it becomes apparent that successful software development organizations in industry have either developed methods that can be modeled or are currently using established models of the software development process. A study of these methods could reveal a preferred way that might be modeled, or a preferred model that should be used for the management of software development. Although the Cosmos model provides a comprehensive descriptive model of the software development process and is applicable to large software projects, what is needed in industry is the development of a model that has prescriptive capability like that presented in the System Dynamic Management Model for small to medium software projects. This could be accomplished through the study of successful large software projects, and application of previous research in the area of prescriptive modeling of large software development projects.

2. Understanding the types of contractual vehicles that can be used to facilitate efficiency in the software development process.

As demonstrated in the Patriot software development

management method through the Engineering Services Contract, contracts have the capability to provide DoD software development managers with needed flexibility to accomplish efficient software development. Since the problem of effective software development management has been recognized for some time, the probability exists that other DoD projects have devised similar types of contractual vehicles. An understanding of these contracts and how to implement them could benefit future software development projects.

4. *Understanding how the lack of Military Standards within DoD software development could affect future software development programs.*

Without a set of military standards combined with the current lack of industry understanding of software development management, it appears that a potential exists for software to become an even greater unknown quantity in weapon system development. By no longer requiring CDRLs, specific plans and documents, and reviews, military managers might not receive the necessary information required to ensure even current management levels for software development projects are maintained.

5. *Understanding how performance specifications can be used to ensure the acquisition of effective military weapon systems.*

While design specifications tell the manufacturer *how* to create a product to ensure that it performs at a required level, performance specifications state only *what* the user wants the product to accomplish, leaving the *how* of the equation up to the manufacturer. In the past, the acquisition of military equipment has been based on the use of design specifications. The sudden shift to performance specifications as the basis for military acquisition, presents DoD with a dilemma of how best to state these specifications to receive the requested product. A study into this area for specific types of products could present insights that might ease this transition.

LIST OF REFERENCES

1. Defense Systems Management College, Mission Critical Computer Resources Management Guide, Ft. Belvior, VA, 1989.

2. Baker, C. and Silverberg, D., Defense News. Vol. 4, No. 51, (December 18, 1989).

3. Frank, W.C., Critical Issues in Software: A Guide to Software Economics, Strategy, and Profitability. New York, NY: John Wiley and Sons, Inc. 1983.

4. Mills, H.D., "Software Engineering: Retrospect and Prospect." The Twelfth Annual International Computer Software and Applications Conference (COMPSAC), pp. 89-96, October 5-7, 1988.

5. Schlender, B.R., "How to Break the Software Logjam." Fortune, pp. 100-112, September 25, 1989.

6. Thayer, R.H., et al., "Major Issues in Software Engineering Project Management." IEEE Transactions of Software Engineering, Vol. SE-7, No. 4, (July 1981).

7. Thayer, R.H., "Modeling a Software Engineering Project Management System." Unpublished Ph.D. dissertation, University of California, Santa Barbara, CA, 1979.

8. Gehring, P.F. Jr. and Pooch, V.W., "Software Development Management." Data Management, pp. 14-38, February 1977.

9. Moore, J.H., "A Framework for MIS Software Development Projects." MIS Quarterly, Vol. 3, No. 1, pp.29-38, (March 1979).

10. Department of Defense, Strategy for a DoD Software Initiative, 1982.

11. Finkelstein, A., "Not Waving but Drowning: Representation Schemes for Modeling Software Development." 11th International Conference on Software Engineering, Pittsburgh, PA, pp. 402-403, (May 15-18 1989).

12. McKeen, J.D., "Successful Development Strategies for Business Application Systems." MIS Quarterly, Vol. 7, No. 3, (September 1983).

13. Basili, V.R., "Improving Methodology and Productivity Through Practical Measurement." A Lecture at the Wang Institute of Graduate Studies, Lowell, MA, (November 1987).

14. Humphrey, W.S., Managing the Software Process, Reading, MA: Addison-Wesley Publishing Company, 1989.

15. Reifer, D.J., Software Management, Los Alamitos, CA: IEEE Computer Society Press, pp. 2-8, 1993.

16. Abdel-Hamid, T.K. and Modnick, S.E., Software Development Dynamics: An Integrated Approach, Englewood Cliffs, N.J.: Prentice-Hall Inc., 1990.

17. Barbucci, M.R., Habermann, A.N. and Shaw, M., "The Software Engineering Institute: Bridging Practice and Potential." IEEE Software, pp. 4-21, 1985.

18. Boehm, B.W., "Software Engineering." Software Engineering. Edited by H. Freeman and P.M. Lewis II. New York, NY: Academic Press Inc., 1980.

19. Merwin, R.E., "Software Management: We Must Find a Way." IEEE, p. 20, 1978.

20. Humphrey, W.S., et al., "The State of Software Engineering , Practice: A Preliminary Report." Proc. 11th International Conference of Software Engineering, pp. 277-288, 1989.

21. Boehm, B.W., "A Spiral Model of Software Development and Enhancement." Computer, Vol. 21, No. 5, pp. 61-72, (May 1988).

22. Basili, V.R. and Musa, J.D., "The Future Engineering of Software: A Management Perspective." Computer, Vol. 24, No. 9, pp. 90-96, (September 1991).

23. Genuchten, M.V., "Why is Software Late? An empirical Study of Reasons for Delay in Software Development.: IEEE Transactions on Software Engineering, Vol. 17, No. 6, pp. 582-590, (June 1991).

24. Yeh, R.T., et al., "A Commonsense Management Model." IEEE Software, Vol. 8, No. 6., pp. 23-33, (November 1991).

25. Marciniak, J.J. and Reifer, D.J., "Software Acquisition Management." Excerpt from Software Acquisition Management, John Wiley and Sons Inc., New York, 1990.

26. Department of Defense, MIL-STD-2167A, Defense Systems Software Development, February 1988.

27. Department of Defense, MIL-STD-498, Defense Systems Software Development, December 1994.

28. Brooks, F.P. Jr., The Mythical Man-Month, Reading. MA: Addison Wesley Publishing Co., 1978.

29. Weeks, P., "Patriot-ATM Defense System." 1993 ADA Year Book, pp. 40-43, (December 1993).

30. Raytheon, Software Development Plan for the Patriot System Vol. I and Vol. II, Bedford, MA, 1993.

31. Department of the Army, Field Manual 44-85, Patriot Battalion Operations, September 1994.

32. Gustine, Col., "Patriot Software Briefing", Naval Postgraduate School, 15 June 1994.

33. Interview between Mr. H. Brown, Chief Financial Division for PPO, Huntsville, AL and author, 18 July 1994.

34. Interview between Mr. A.Q. Oldacre, Deputy PEO Ballistic Missile Defense, Huntsville, AL and author, 18 July 1994.

35. Moore, L.E., Patriot ADA Systems Implementation Plan Memorandum, PPO, Redstone, AL, 28 November 1989.

36. Public Law, 102-396, Sec. 9070, October 1990.

37. Department of Defense, DoD Directive 5000.1, DoD Acquisition Policy, 1991.

38. Department of Defense, DoD Instruction 5000.2, DoD Acquisition Policy, 1991.

39. Department of Defense, DoD Instruction 8120.2, "Automated Information Management Systems", 1992.

40. Department of Defense, DoD Directive 3405.1, "Software Management", 1994.

41. Department of Defense, MIL-STD-2168, "Software Quality Program", 1988.

42. Department of Defense, "Implementing Specifications and Standards Reform" Memorandum, Office of the Secretary of Defense, 1994.

43. Paul, R.A., Metrics to Improve the U.S. Army Software Development Process, U.S. Army Software Test and Evaluation Panel (STEP), 1992.

44. Gustine, Col., "Patriot Software: A Gulf War AAR Briefing", PPO, Redstone, AL, 6 August 1991.

45. Hughes Aircraft, "Software Development Briefing", Naval Postgraduate School, 16 February 1995.

46. General Accounting Office, Report IMTEC-92-62-BR Embedded Computer Systems: Defense Does Not Know How Much it Spends on Software, USGPO, July 1992.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5101	2
3.	Professor David V. Lamm, Code SM/Lt Department of Systems Management Naval Postgraduate School Monterey, California 93943-5000	4
4.	LTC Keith Snider, Code SM/Sk Department of Systems Management Naval Postgraduate School Monterey, California 93943-5000	3
5.	Professor Tarek Abdel-Hamid, Code SM/Al Department of Systems Management Naval Postgraduate School Monterey, California 93943-5000	1
6.	Professor Marty McCaffrey, Code SM/Mf Department of Systems Management Naval Postgraduate School Monterey, California 93943-5000	1
7.	LTC John T. Dillard, Code SM/Dj Department of Systems Management Naval Postgraduate School Monterey, California 93943-5000	1
8.	OSDA (RDA) ATTN: SARD-ZAC 103 Army Pentagon Washington, D.C. 20310-0103	1
9.	S. G. Drake 5136 Temple Court El Paso, Texas 79924	2
10.	Defense Logistics Studies Information Exchange U.S. Army Logistics Management College Fort Lee, Virginia 23801-6043	1
11.	PATRIOT Project Office ATTN: Project Manager P.O. Box 1500 Huntsville, Alabama 35807-3801	1