A guide to selecting software metrics for the acquisition of weapon systems

Ramgolam, Rakhee H.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A GUIDE TO SELECTING SOFTWARE METRICS FOR THE ACQUISITION OF WEAPON SYSTEMS
by

Rakhee H Ramgolam

September 2001

Thesis Advisor:	Brad Naegle
Associate Advisors:	Mark E. Nissen
	Luqi

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 01	Master's Thesis

4. TITLE AND SUBTITLE: A Guide to Selecting Software Metrics for the Acquisition of Weapon Systems

5. FUNDING NUMBERS

6. AUTHOR(S) Rakhee H Ramgolam

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)
N/A

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)***

Modernization of Department of Defense (DoD) weapon systems has resulted in an ever-increasing dependence on software. Despite technological advances in the software field, software development remains costly and one of the highest risk factors on most weapon system programs. The use of software metrics is a methodology for mitigating this uncertainty so that software development progresses under informed decision making. Software metrics are essential tracking tools used by program managers to monitor and control risk areas. However, the choice of metrics for a program is critical to their usefulness. This research provides a guide to acquisition managers on selecting the most effective metrics to use in management of weapon system software. The study identifies key issues in the use of software metrics experienced by program managers. The study recommends a revised set of metrics and improvements to the use of metrics based on innovations and improvements in the software field as well as software estimation tools that facilitate the use of these software metrics.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Software, Metrics, Program Management, Acquisition, Weapon Systems, Software Life Cycle	95
	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

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

A GUIDE TO SELECTING SOFTWARE METRICS FOR THE ACQUISITION OF WEAPON SYSTEMS

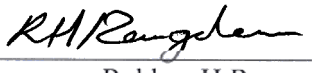
Rakhee H Ramgolam
DoD Civilian, Arms Corporation of South Africa
B.Sc. in Engineering (Chemical), University of Durban-Westville, 1996

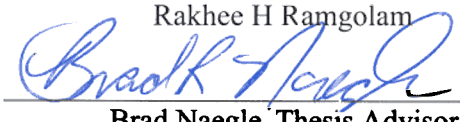
Submitted in partial fulfillment of the
requirements for the degrees of

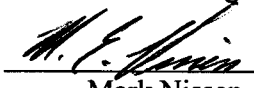
MASTER OF SCIENCE IN MANAGEMENT
AND
MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

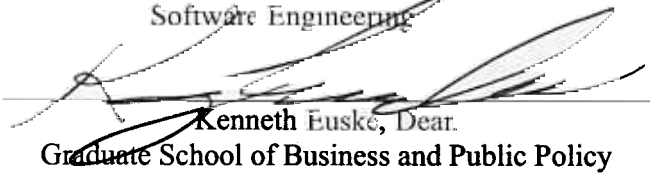
NAVAL POSTGRADUATE SCHOOL
September 2001

Author: 
Rakhee H Ramgolam

Approved by: 
Brad Naegle, Thesis Advisor


Mark Nissen, Associate Advisor

Luqi, Associate Advisor and Chair
Software Engineering


Kenneth Euske, Dean
Graduate School of Business and Public Policy

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Modernization of Department of Defense (DoD) weapon systems has resulted in an ever-increasing dependence on software. Despite technological advances in the software field, software development remains costly and one of the highest risk factors on most weapon system programs. The use of software metrics is a methodology for mitigating this uncertainty so that software development progresses under informed decision making. Software metrics are essential tracking tools used by program managers to monitor and control risk areas. However, the choice of metrics for a program is critical to their usefulness. This research provides a guide to acquisition managers on selecting the most effective metrics to use in management of weapon system software. The study identifies key issues in the use of software metrics experienced by program managers. The study recommends a revised set of metrics and improvements to the use of metrics based on innovations and improvements in the software field as well as software estimation tools that facilitate the use of these software metrics.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS

AIS	Automated Information System
ARM	Automated Requirement Measurement
CRU	Computer resource utilization
CSC	Computer software component
CSCI	Computer software configuration item
CSU	Computer software unit
DoD	Department of Defense
DAU	Defense Acquisition University
EVM	Earned Value Management
FP	Function points
HWCI	Hardware configuration item
LOC	Lines of code
MIL-STD	Military Standard
PM	Program manager
SLOC	Source lines of code
US	United States

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Col John Dillard, Mr Art Santa Donato, Col Dave Matthews and Col Rodney Tudor for their insightful interviews and kind cooperation.

I would like to thank my advisors, Col Brad Naegle for his support, Dr Mark Nissen for his timely feedback and Dr Luqi for her support and constant encouragement.

Lastly, I would also like to thank my husband Rajesh Ramgolam for proofreading and editing this thesis, and for his unfailing support in all my endeavors.

I. INTRODUCTION

A. PURPOSE

The purpose of this thesis is to analyze the use of software metrics for the acquisition management of systems software. The focus is primarily on program managers' use of software metrics as a management tool to identify issues and challenges. An analysis of the issues and challenges is conducted to identify software metric innovations currently practiced in the field of software engineering that could improve the use of software metrics as a program management tool. The objective of this thesis is to serve as a guide in selecting software metrics for the acquisition of software intensive systems. The thesis also examines estimation tools that facilitate the use of these metrics.

B. BACKGROUND

Modernization of Department of Defense (DoD) weapon systems has resulted in an ever-increasing dependence on software. Despite technological advances in the software field, software development remains costly and one of the highest risk factors in most weapon system programs.

The use of software metrics is a methodology for mitigating this uncertainty so that software development progresses under informed decision making. Software metrics are essential tracking tools used by program managers to monitor and control risk areas. However, the choice of metrics for a program is critical to their usefulness. The purpose of this research is to provide a guide to acquisition managers on selecting the most effective metrics to use in management of system software. The study identifies what software metrics are being used by program managers and analyzes the effectiveness of these metrics. The study further recommends an improved set of metrics based on innovations and improvements as well as software estimation tools that facilitate the use of these software metrics.

C. RESEARCH QUESTIONS

1. Primary Research Question:

i. What software metrics are suitable for the acquisition management of weapon systems software?

2. Secondary Research Questions:

i. What is the level of program managers' awareness of software metrics as a management tool for system software?

ii. What software metrics are currently used in the acquisition of major weapon systems and how effective are they?

iii. What other software metrics could be used to improve system software management?

iv. What software metrics are appropriate for different phases of the acquisition process?

v. What estimation tools facilitate the use of these software metrics?

D. SCOPE OF THESIS

This research focuses on software management on software-intensive weapon systems by program managers in the DoD environment. DoD program managers of software-intensive weapon systems are interviewed to investigate the use of software metrics as a risk mitigation tool.

Software metrics innovations and improvements conducted for governmental, academic and commercial purposes are reviewed to identify effective metrics appropriate for use by program managers in the DoD. The results of the analysis provide insight into those software metrics most suitable for management of system software by DoD program managers for overall system software management, and those software metrics useful for specific phases of the software development cycle.

Research will demonstrate the use of software metrics in the software development process. However the research on software metrics is confined to those relevant for use by program managers in the DoD environment, rather than management of software development in general.

E. METHODOLOGY

The methodology used in this thesis research consists of the following steps:

i. Interview program managers to determine their use of software metrics for the management of weapon system software and how effective these metrics prove.

ii. Determine what other software metrics could be used for the management of weapon system software by conducting a literature review of innovations and improvements to software metrics.

iii. Determine what estimation tools are available to facilitate the use of these software metrics by conducting a literature review of software estimating tools.

iv. Identify metrics most suitable for the management of weapon systems throughout the acquisition process and, metrics useful for specific phases of the software development cycle by analyzing literature and interview responses.

F. ORGANIZATION

Chapter II provides an overview of software management in the acquisition environment. It includes the DoD acquisition process, software development within this process, and the types and role of software metrics in software development.

Chapter III discusses the use of software metrics by program managers in the DoD acquisition environment and how effective they were.

Chapter IV is an analysis of the common software metrics used for weapon systems software management and an identification of other metrics suitable for weapon systems software management. Estimation tools that facilitate the use of these metrics are also discussed.

Chapter V summarizes the findings and addresses the research questions. Then it concludes with recommendations resulting from the analysis and identifies areas for further study.

G. BENEFITS OF THE STUDY

This study will serve as a guideline for acquisition program managers in selecting software metrics for the management of weapon system software. Common software related issues and challenges experienced by program managers are identified and software metric effectiveness is assessed.

H. SUMMARY

This chapter identified the major thesis topic, outlined the research methodology, scope of research, and organization of the thesis. The next chapter provides background information necessary for understanding the data, analyses, recommendations and conclusions presented.

II. SOFTWARE MANAGEMENT IN THE ACQUISITION ENVIRONMENT

A. INTRODUCTION

This chapter describes Department of Defense (DoD) use of software, the DoD acquisition process, software development activities, the role of software metrics in these activities and the types of metrics.

In particular, the importance of software in DoD weapon systems, which is illustrated by the DoD's reliance on software, is examined to motivate the need for this research work, which is to improve software management in the DoD. Next the DoD software acquisition environment and software development risks are described to understand the role of software metrics in improving software acquisition management.

B. USE OF SOFTWARE BY THE DOD

The role of military software was aptly described by the 1987 Report of the Defense Science Board Task Force on Military Software [Ref 1] as follows:

> Software plays a major role in today's weapon systems. The "smarts" of smart weapons are provided by software. Software is crucial to intelligence, communications, command, and control….Software provides a major component of U.S. war-fighting capability.

In fact, software provides the technological edge to any advanced military. Software is the essential component in modern weapons and has forever changed the military's concept of the battlefield, the world over.

Software increases the capabilities of war fighters by arming them with powerful, "smart" weapons and decision support tools. Using a single aircraft with a smart bomb to attack a target carefully selected from appropriate intelligence information costs much less than sending in multiple aircraft with multiple conventional weapons to neutralize a minimally defined target area [Ref 2]. Software based systems will continue to receive an increasing share of defense budgets because they allow weapon systems to be more flexible and effective at an overall cost.

The U.S. DoD's [Ref 3] reliance on software-intensive systems is illustrated by the fact that it has over:

i. 1.5 million computers (of which 827, 000 are personal computers),

ii. 28, 000 software systems (of which 11% are mission-critical),

iii. 10, 000 computer networks,

iv. 88, 000 communications systems, and

v. 100,000 facility support systems (e.g., security and medical support systems).

In 1992, the DoD has estimated that it spends $24 billion to $32 billion annually for software embedded in weapon systems [Ref 4]. This amount was approximately 8-11% of the DoD budget for that year. In the next 15 years, it is estimated that software may increase to an annual cost of $50 billion and account for up to 20% of the DoD budget [Ref 5].

The two major DoD software domains are Weapon System Software and Automated Information System (AIS) software, which are divided into subcategories as shown in Figure 1. Software Technology Support Center [Ref 2] provides a detailed explanation of the subcategories within the domains. Discussion of software management relates to both domains and exceptions are highlighted. However, weapon system software is the focus of the research in this thesis.

C. SOFTWARE ACQUISITION ENVIRONMENT

The software industry is approaching its 50-year mark, yet the problems that plague software acquisition still persist. Studies have shown that many current problems with software–intensive programs are rooted in poor management, which overshadow the technical difficulties [Ref 2].

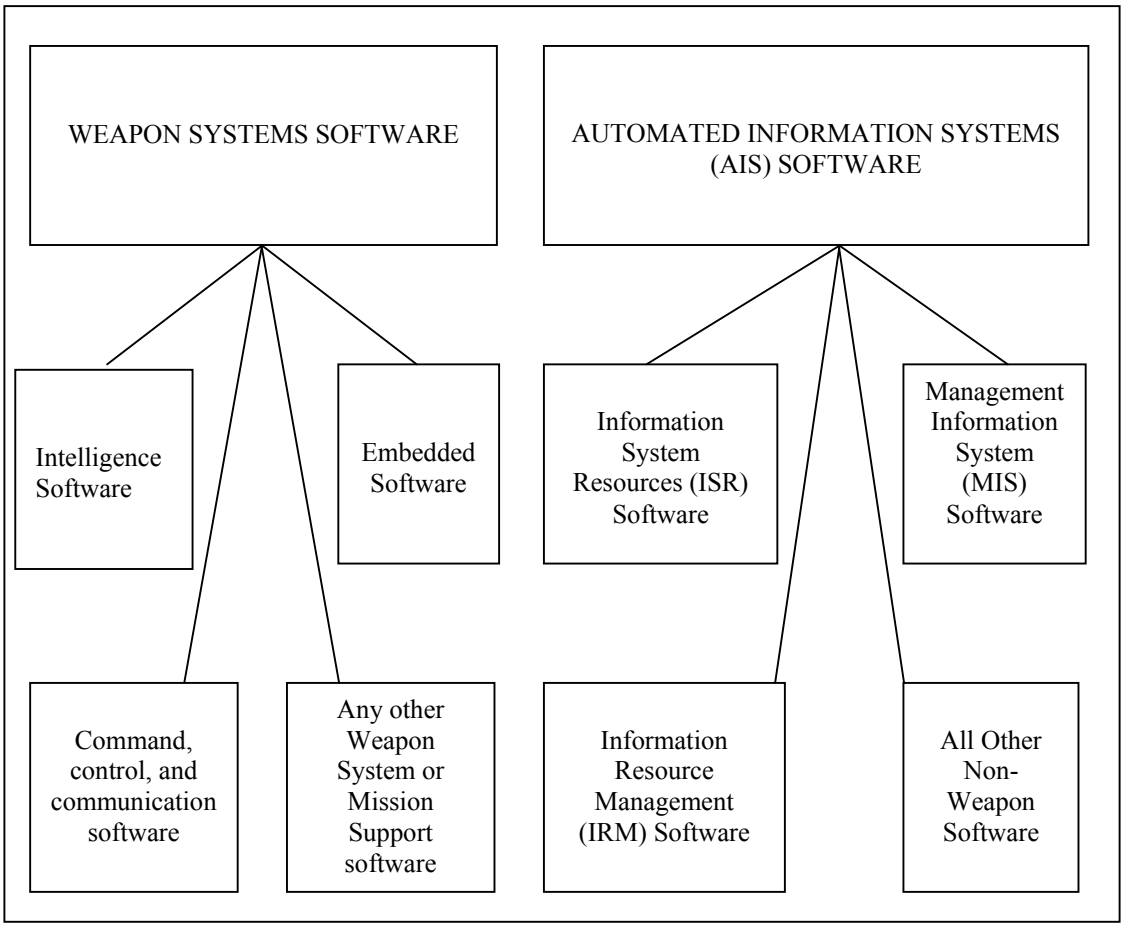


Figure 1: Department of Defense Software Domains. From Ref. [2]

The complexities of software development, coupled with immature software acquisition processes of defense departments, contribute to making software the highest risk element on these programs. Software acquisition management is therefore the greatest challenge currently facing Program Managers (PMs).

According The Software Technology Support Center [Ref 2], software-intensive acquisitions fail due to one or more of the following causes:

i. Technology-driven solutions

ii. Unstable requirements

iii. Software's inherent complexity,

iv. Thinking automated technology will make up for poor engineering practice,

v. Poor estimation of size, schedule, and cost,

vi. Inadequate software staffing, and

vii. The domino effect resulting from any combination of the above.

The success or failure of a major software-intensive acquisition program depends on a highly complex combination of factors, not all of which are under the control of the acquisition manager.

Changing trends in software acquisition is an example of this. Many governments worldwide, such as the United States, have shrunk their defense budgets in efforts to lower the government's cost of doing business. In the United States, the Federal Acquisition Streamlining Act of 1994 mandates this trend to address the costly and inefficient procurement of software-intensive weapon systems. Shrinking procurement funds and an increasing number of missions are compounding the software management challenge for acquisition managers.

The U.S. DoD is the world's single greatest consumer of software and is greatly affected by the software labor shortage. In the U.S. Government salaries are not competitive with those in the private sector, thus exacerbating the labor shortage situation. This shortage results in program delays, cost overruns and poor software quality, all of which compound the acquisition failure factors listed above.

D. SOFTWARE DEVELOPMENT

1. Software Development as Part of System Acquisition

The software acquisition problems discussed above are mostly attributable to poor management of the software development process. It is therefore important to understand the system acquisition context in which software is developed.

The U.S. DoD acquisition process was designed to manage a program through sequential phases. Each phase ends with a major milestone decision, which marks the

completion of a phase of the life cycle and entry into the next based on program progress reported by the Program Manager (PM).

Figure 2 illustrates three life cycles; system acquisition life cycle, weapon system software life cycle and automated information system software life cycle as defined in DoD 5000.2 (Part 3), DoD-STD-2167A, and DoD-8120.2, respectively. DoD-STD-2167A has subsequently been superceded by MIL-STD-498, which improved many key areas of the older standard and applied to Weapon Systems Software as well as AISs. The DoD approved the use of MIL-STD 498 for two years to allow the IEEE and EIA working group, time to develop a new non-government software standard, the J-STD-016 in that time frame. This working group is currently tailoring the ISO 12207, the new international standard, into US 12207, which includes much of the J-STD-016 technical detail.

As shown in Figure 2, weapon system software is a subpart of the system life cycle and follows a parallel, but not always common path with the system life cycle (this is true for the Automated Information System (AIS) software life cycle).

Each activity in the weapon system software life cycle involves the generation of selected documents and requires reviews, audits, and inspections as described below:

i. System requirements and design: During this phase, hardware configuration item (HWCI) and computer software configuration item (CSCI) requirements are allocated. Other activities include requirement refinement, operational concept analysis, tradeoffs (e.g., optimization of alternative solutions and minimizing constraints), and risk evaluation.

ii. Software requirements analysis: Software requirements are baselined and documented. Prototypes of higher risk computer software configuration items are built and demonstrated.

iii. Preliminary design: Computer software configuration item requirements are allocated to computer software units (CSUs) for implementation. Instructions for the programmer are documented. In addition, system integration test procedures are designed.

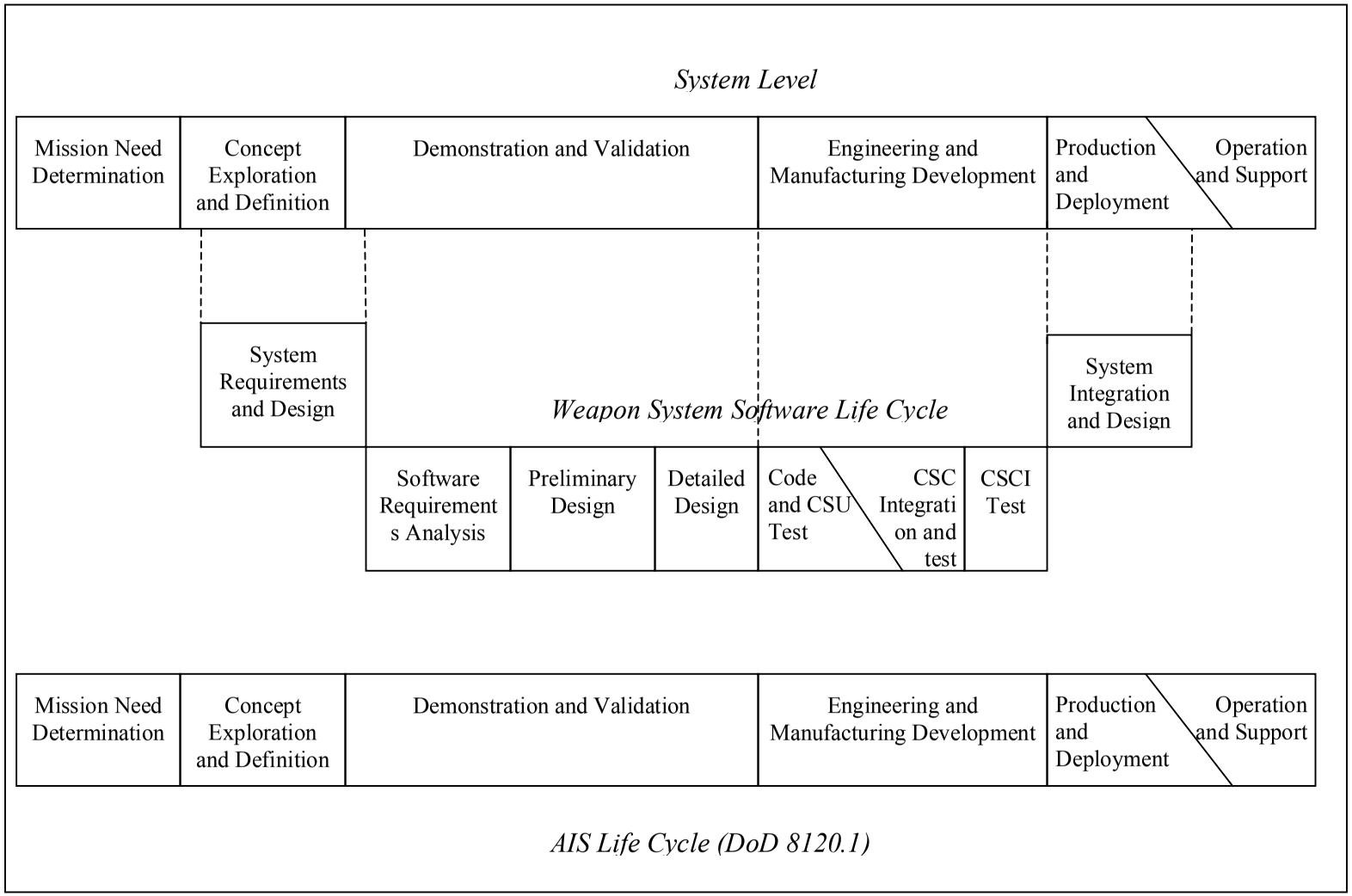


Figure 2: Lifecycles Compared. From Ref [7]

iv. Detailed design: This phase defines software structure. Software is partitioned into modules. Reusable software is identified. This phase includes the finalization of the software architecture.

v. Code and CSU Test: Code is written for each computer software component (CSC), module, and CSU. Unit testing is performed to detect coding errors.

vi. CSC Integration and Test: CSUs are integrated into CSCs and tested.

vii. CSCI Test: The fully implemented CSCI is tested to check if it satisfies software requirements specifications and software interface specifications.

viii. System Integration and Test: Entire weapon system is tested to determine if it meets system-level requirements, and performance of the software in the operational environment. Pending a successful outcome of a Formal Qualification Review, the Government may accept the system.

2. Software Development in Commercial Applications

In the 1980s, the DoD was by far the largest consumer of software products. With the advent of information age, software engineering research for commercial applications driven by commercial use of software has increased exponentially.

The previous section describes software development in the context of system acquisition. This view is adapted from the software life cycle in the software engineering field. Thus a description of software development in software engineering has many similarities and another software development description in the software engineering context may seem redundant. However, it is necessary to describe software development by developers of software other than for DoD use due to the growing consumption of software outside the defense environment. Further motivation for describing software development in the software engineering environment is the use of innovations in the field of software engineering that this research work addresses to improve software management in the DoD.

The waterfall model, a commonly accepted model of the software life cycle, is described to reinforce the view of the software development process from the development personnel perspective, who have a view more focused on the technical aspect than on the management. The model is described in detail by Boehm [Ref[6]. The essential phases of this software life cycle are described below.

i. Requirements and Specifications – This phase should produce a complete specification of the required functions and performance characteristics of the software. It should also address the question of resource needs and preliminary budget estimates.

ii. Product design – This phase should specify the overall system configuration, the implementation language, major modules and their interfaces, configuration control and data structures and a testing plan.

iii. Detailed Design - This phase should produce more detailed module specifications including their expected size, the necessary communication among modules, algorithms to be used, interface stat structures, and internal control structures. It should highlight important constraints relative to timing or storage, and include a plan for testing the individual modules.

iv. Programming/Coding – This phase should produce an implementation of the modules in the chosen language together with unit testing and subsystem testing.

v. System Integration – This phase, usually completed by a group independent of the original analysts and programmers, should subject the integrated modules to extensive testing to ensure that all functional requirements are met. Errors are corrected as discovered.

vi. Installation/Acceptance – This phase should deliver the product to the user organization for final acceptance tests within the operational environment for which it is intended. Documentation and user manuals are delivered,

training is conducted, problem reports are recorded, and corrections made until the customer accepts the product.

vii. Maintenance - This is a continuing phase in which additional discovered errors are corrected, changes in code and manuals are made, new functions are added, and old functions are deleted.

E. SOFTWARE METRICS

1. The Role of Software Metrics in Software Development

Traditional engineering, such as chemical, civil, mechanical and electrical, has always relied heavily on metrics, measurement and models to help arrive at quantitative assessment of cost and quality during the development of a product. However, system engineering, until recently, has been perceived to be impervious to these engineering approaches with software development viewed as an art rather than a science.

With the current trend of costly, untimely and unreliable software products, this lack of quantitative assessment is a major cause of the problems currently experienced in software development [Ref. 8]. Thus, a growing realization among managers at all levels is that a more disciplined approach to software development must be adopted.

As software development has evolved, software metrics have become the primary tool available to control and manage software projects. However, the acceptance of metrics was slow to take place since software developers and others were slow to change their view of software development. Finally, after many years of failed software development efforts, software development has come to be viewed as a science and the use of metrics has become widespread [Ref 9].

The proper use of measurement, models and metrics can facilitate such a disciplined approach to software development. The use of metrics is therefore essential to the successful management of development and maintenance of a cost-effective, timely and reliable product, which is the role it currently fulfills in other engineering disciplines.

2. Types of Software Metrics

Software is developed for a wide variety of applications and in many different languages. Thus, there is a wide range of metrics. Metrics are often classified as either process or product metrics [Ref 8].

i. Process Metrics: quantify attributes of the development process and of the development environment (e.g., resource metrics such as the experience of programmers), as well as the cost of development and maintenance.

ii. Product Metrics: measures of the software product (e.g., size of the product, efficiency, reliability, etc.).

It may be difficult to classify metrics as strictly process or product metrics. For example the number of defects discovered during formal testing depends both on the product (the number of code segments that are erroneous), and on the process used in the testing phase (the extensiveness of the testing).

Typical metrics used for management of software development include size, quality, complexity, requirements, effort, productivity, cost, schedule and others. The following sections in this chapter present some product and process metrics used in the software engineering field that possess qualities that qualify them as potential candidates for use by PMs. The product metrics are subcategorized as size, data structure and logic structure metrics and an example of each is discussed.

a. Size Metrics

Programs are written in numerous languages that give them different characteristics, but the common characteristic that all programs share is size. There are many possibilities for representing the size of a program (e.g., source lines of code). This metric is useful as:

i. It is easy to compute after the program is completed.

ii. It is the most important factor for many models of software development.

iii. Productivity is normally based on a size measure.

The principal size measure found to be the most useful in characterizing software is lines of code [Ref 8].

The lines of code metric are evaluated by counting the lines of code in a program. Although this may seem to be a simple metric that can be counted algorithmically, there is no general agreement about what constitutes a line of code. Most researchers agree that the line of code measure should not include comments or blank lines as this may encourage programmers to introduce, artificially, many such lines in project development in order to create the illusion of high productivity, which is normally measured in Lines of Code/Person-Month (LOC/PM). An accepted definition of lines of code used by researchers today is any line of program text that is not a comment or blank line [Ref 8].

A disadvantage of using this metric is its application to programs written in languages that provide different levels of functionality. For example, a program written in Basic will require more lines of code than if C++ is used. In this case the higher number of lines of code gives the illusion of higher productivity when, in fact, the opposite is true.

A further shortcoming is that this metric can only be evaluated after the programming language is selected, so that this metric is limited to later stages of software development.

b. Data Structure Metrics

The purpose of any software is to process data. Some data are input to a system, program, or module; some data may be used only internally; and some data are the output from a system, program or module. Thus data structure metrics, defined as a count to the amount of data input to, processed in, and output from software, is an important software metric.

An example of a data structure metric that is also a size metric is function points. Function points (FP) measure the size of a software project's work output or work product, rather than measure internal features such as lines of code. FPs evaluates the size of the functional user interfaces that are supported or delivered by the software. In simple

terms, FPs measure what the software must do from an external, user perspective, irrespective of how the software is constructed.

Function points are calculated as the weighted sum of factors that relate to user requirements: External Inputs, External Outputs, External Inquiries, External Files (interfaces to other systems), and Internal Files [Ref 10, p.639-640]. Thus, function points can be estimated using the software requirement statements.

Function Points have the advantage of independence over the physical implementation and languages used to develop the software and they remain consistent no matter what development language or technique is used. In fact, once estimates for function points are calculated, they can be used to estimate LOC. Function points is a useful metric for software managers because:

i. It is a metric that can be estimated early in development. It can be calculated after the preliminary analysis of the system.

ii. It is easy to calculate. There are only five input parameters to compute and fourteen system-dependent adjustments, but the whole process can be done manually.

iii. It is the first metric that relates complexity of software to number of lines of code.

The main disadvantage of the function points metric is that there are not extensive databases on which wide-scale use of function points is dependent. A comparison of LOC and Function Points is illustrated in Table 1 below.

c. Logic Structure Metrics

The logic structure of a program allows it to perform different operations dependent upon different input data or intermediate calculations. In a study in which experts rated complexity metrics, eight of nine product metrics considered important were related to measures of a program's logic structure (the ninth important metric was related to data). There is no agreement on which logic metric is the most important, and a survey of logic metrics showed counting methods to be at a very technical level. However, logic structure metrics are important to be considered for use in the

management of software development as they account for characteristics in software development that are not captured by size or data structure metrics.

FUNCTION POINTS	SOURCE LINES OF CODE
Specification-based	Analogy-based
Language independent	Language dependent
User-oriented	Design-oriented
Variations a function of counting conventions	Variations a function of languages
Expandable to source lines of code	Convertible to function points

Table 1: Comparison of Function Points and Lines of Code. From Ref. [2, Ch. 8, p.33]

McCabe's Cyclomatic Complexity Metrics, a well-known logic structure metric, is useful to software development because it is closely related to design errors and defects. Complexity metrics, therefore, can predict impacts to quality. The quality of the product can only be determined at the end of the development process. It is necessary to measure complexity as a predictor of quality. The collection and interpretation of data for use of this metric requires technical expertise. In particular, this metric focuses on the number of linearly independent paths (structure) through a program. The tedious nature of information collection for this metric is expedited by the use of automated tools [Ref 9].

F. CHAPTER SUMMARY

Software is the essential component in modern weapons and has forever changed the military's concept of the battlefield. However, in spite of the software industry approaching its 50-year mark, the problems that plague software acquisition still persist. Studies have shown that many current problems with software–intensive programs are

rooted in poor management rather than in technical difficulties. Finally, after many years of failed software development efforts, the software development process has evolved and software metrics have become the primary tool available to control and manage commercial software projects. The use of software metrics in the acquisition environment, however, is lagging the private sector, and warrants a study for the application of metrics research that aids commercial software development.

III. SOFTWARE METRICS IN THE ACQUISITON ENVIRONMENT

A. INTRODUCTION

This chapter examines the use of software metrics by PMs in the acquisition of software-intensive weapon systems, which serves as the data set for later analysis. First, the past and current use of software metrics by the DoD, as mandated in US policies and guidelines, is discussed. The next section explains the methodology used in the collection of software metric data. Next, these data are presented under three themes in which the use of software metrics by program managers of software intensive systems is categorized. The three themes are:

i. The level of awareness and use of software metrics by PMs.

ii. Metrics used.

iii. The metrics-phase coupling process of choosing metrics corresponding to software development phases.

B. DOD USE OF SOFTWARE METRICS – PAST AND PRESENT

Prior to 1990, policy specifically targeting the management of software-intensive programs was largely lacking in the US Department of Defense, as was the case the world over. This resulted in a long line of problem-plagued programs. The U.S Defense Science Board Task Force and the U.S. Army's Software Test and Evaluation Panel (STEP) confirmed this lack of focus in 1987 and 1990 respectively [Ref 11]. As a result of their findings, the STEP made a number of recommendations to the Army on how to address the software problem. One of these recommendations was the enforcement of a specific set of software metrics, which became known as the Army "STEP metrics". Specific requirements on the use of these metrics were defined in DA Pamphlet 73-1. However, because of the rigid nature of these requirements, they were subject to much criticism within the acquisition community, especially among program managers who thought the policy was unnecessarily restrictive. With the outset of acquisition reform

initiated by the U.S. Secretary of Defense, the requirements were relaxed in 1994. Program managers of software-intensive systems were still required to report the STEP metrics, but they were given the flexibility to determine which data elements would define the 12 metrics. In March 1996, a new set of acquisition reform initiatives were set forth by the new DoD 5000 series guidance. The DoD 5000.2 regulation shifted the focus from a specific set of metrics to the following six management issues:

i. Schedule and Progress – regarding completion of program milestones, significant events, and individual work items.

ii. Growth and Stability – regarding stability of required functionality or capability and the volume of software delivered to provide required capability.

iii. Funding and Personnel Resources – regarding the balance between work to be performed and resources assigned and used.

iv. Product Quality – regarding the ability of the delivered product to support the user's need without failure, and problems and errors discovered during testing that result in the need for rework.

v. Software Development Performance – regarding the developer's productivity capabilities relative to program needs.

vi. Technical Adequacy - regarding software reuse and use of approved standard data elements, and compliance with the DoD Joint Technical Architecture (JTA). [Ref. 2, App.V]

The policy described above remains in force today with the addition of one more management issue added in March 1998; - Program Success - that requires strategic goals to be linked to the achievement of performance measures. The experience of program managers with the current policies on software management and software metrics is the subject of the next section.

C. METHODOLOGY USED IN DATA COLLECTION

The following methodology is used in the collection of data presented in Appendix B and summarized below.

i. Initial contact was made with former PMs working in the Graduate School of Business and Public Policy to identify suitable candidates who had sufficient experience with the management of software-intensive weapon systems.

ii. Selected PMs were interviewed, which was the central activity of the data collection process. The interviews were based on a series of questions designed to elicit information on the PM's experience with the use of software metrics during the acquisition of software-intensive weapon systems. The three themes of questions asked during the interviews were the level of awareness and use of software metrics by PMs, what metrics were used and the metrics coupling process of choosing metrics corresponding to current software development phases. Appendix B contains the interview questions and responses.

iii. The last step involved collating and analyzing the interview responses to identify issues commonly experienced by the PMs interviewed. The results are presented in the following section.

D. THE USE OF METRICS FOR ACQUISITION OF SOFTWARE-INTENSIVE MAJOR WEAPON SYSTEMS

1. Level of Awareness and Use of Software Metrics by Program Managers

Most Program Managers (PMs) interviewed used software managerial metrics and found they provided great utility to the management of software. However the metrics used had to be the correct metrics, specific to the system being acquired. Therefore, the 1993 edict by the Army mandating the use of the STEP metrics was viewed as a "one size fits all" mindset, which faced resistance from PMs.

A second reason for this resistance was the lack of additional resources allocated for the collection of metric data. This brings out the cost implications associated with the use of software metrics. Some PMs agreed that these costs were outweighed by the support metrics provided the PM in his/her decision-making.

Metrics were commonly used in the program management plan and the risk management plan. Metrics were used more as a tracking tool for assurance on risk areas, rather than a tool used directly for program management planning purposes. No PMs interviewed used metrics during the planning stage where critical decisions are made. Even during phases after planning, they felt that the metrics used were not sufficient to track the entire software development.

The PMs interviewed were very reliant on their contractors' expertise in the development of software. This included the development of metrics and collection of metric data. This was in large part because of a lack of software expertise on the part of the PM. PMs adopted a management by exception approach to software, and cited competing priorities for this management style. This hands-off approach to software management was also enabled by the PMs designating personnel from the program office for management of software. In one case, software management was subcontracted to MITRE, a corporation that is a federally funded non-profit research contractor.

Most PMs received training in the form of software acquisition courses. PMs who attended these courses felt that the material covered was not substantial enough to give them confidence to independently manage software. These courses explained what software metrics are, but PMs were not informed of their potential use and development. One PM did not receive any training in software acquisition. PMs that did receive some training in software acquisition showed some interest in the innovations and improvements made in the field of software metrics. However, when questioned if they would consider the use of these improvements in their programs, PMs said they still prefer assigning metric development and use to the contractors.

2. Metrics Used in the Management of Software Acquisition

Most PMs used traditional metrics to track cost and schedule, which are commonly used for hardware, such as Earned Value Management (EVM) to track costs and schedule charts to monitor schedule. The most common technical performance metrics used were Source Lines of Code (SLOC) and computer resource utilization. The metrics, and their use by the PMs interviewed, is explained in the sections below.

a. Schedule

This metric tracks the progress of the software development in terms of schedule. The metric was measured from planned versus actual schedule, which could be elicited from schedule charts. Thus development of software had to be tracked using schedule charts, typically Gantt charts, for each hardware and software configuration item as well as the master program schedule, which depicted milestones across time. When the contractor did not manipulate charts, slippages in schedule were visible. The problem of misrepresentation of schedule charts by the contractor was eliminated, to some extent, by the use of color-coding that highlighted slippage and by making the display of original baselines mandatory on these charts. PMs found the graphical representation of schedule metrics easy to use.

b. Earned Value Management

The most common cost metric used was the Earned Value Management system that is required by the US Cost Accounting Standards report. The metric used to monitor costs was the planned versus actual costs. The information needed to determine the metrics were extracted from the monthly reports.

Costs were decomposed into the lowest cost account management level and an engineer was assigned as cost account manager for CSCIs that were functionally decomposed in the work breakdown structure. Monthly reports showed the roll-up of total and subtotals of major segments of the program. This roll-up was particularly important where multiple contractors were used, as it provided the PM a consolidated view of the costs.

c. System Resource Utilization

The development of the software product was tracked using system characteristics such as process size, virtual memory usage, file size, process execution time, central processing unit utilization and queue sizes. This metric was usually collected during system-level stress testing where it served as a performance metric to indicate the degree to which the software product was changing or approaching design limits.

d. Manpower

The manpower metric consisted of two parts. The first part was a measure of effort that is usually determined by planned versus actual labor hours. This provided an indication of the productivity of the human resources assigned by the contractor. The second part was the quantity and types of personnel required for the development. PMs used this information to determine if the contractor had scheduled an adequate staffing level to produce the product with the time and budget allotted.

e. Breadth of Testing

The breadth of testing metric was depicted by the number of system specifications successfully tested compared to the number of system requirements tested. This metric was used as a performance metric to show the degree to which required functionality was successfully demonstrated and the amount of testing performed on new functionality. Breadth of testing is also known as "black-box" testing that evaluates performance based on obtaining correct outputs to prescribed inputs. This test, however, does not show that the current software system is free of errors, as this status is virtually unobtainable and not economically feasible.

f. Requirements Stability

This metric tracked the number of system requirements added, changed and deleted. The objective of this metric is to show the amount of change made to the system and software requirements as the development progresses.

g. Requirements Traceability

The requirements traceability metric traces the number of system requirements to CSUs implementing system requirements. This metric ensures that all requirements are implemented (to check for completeness of the system specifications) and that all CSUs implement system requirements (to identify unnecessary CSUs).

h. Fault Profiles

The fault profiles metric is a performance metric that provides insight into software quality, maturity, and problem correction rate. It is measured by counting the number of errors identified, and corrected during testing.

i. Number of Defects

The number of defects metric was a common metric used during the Engineering Manufacturing Development (EMD) phase. It is measured by the number of defects identified during testing.

3. Metric to Development Phases Coupling

This section reports PMs' experience of metrics use during different phases of software development. Most PMs interviewed were assigned to software-intensive programs during Phase III of the acquisition process as stipulated in the unrevised U.S. DoD 5000 series. This is the Engineering and Manufacturing Development (EMD) phase.

Most PMs did not use metrics early in the acquisition. They indicated that, at this stage, system requirements were not sufficiently understood to use metrics. In addition, due to requirements creep, which is more common in the early phases, PMs did not want to commit resources to obtain estimates of a changing system. In one case, the requirements traceability metric was used during the early phases. One PM cited using computer resource utilization as a useful metric to monitor the effects of requirements creep. This metric measures process capability to ensure that the system is not overburdened by the addition of functionality.

Software metrics, however, are required to be developed early in the acquisition cycle so that software development monitoring and reporting can be included in the contract. Metrics are required to be included in the statement of work prior to the prototyping phase and prior development.

Schedule, EVM and CRU metrics were used throughout the acquisition cycle. The manpower metric was used, following the concept and design phases when the management of personnel overhead takes on more significance. It was also used as a guide to evaluate the adequacy of the contractor staffing levels during the early phases. However, this estimate was also plagued by the requirements creep changes discussed above, that lower the value of initial estimates.

Breadth of testing, fault profiles, number of defects and other defects metrics are commonly used during the EMD phase. However, even during these later phases when metrics were used more readily, PMs indicated that software metrics alone were not sufficient to track the entire software development.

E. CHAPTER SUMMARY

This chapter presented software metric data obtained from interviews with PMs that managed software-intensive major weapon systems. The data are presented under three themes:

i. The level of awareness and use of software metrics by PMs.

ii. The effectiveness of the metrics used.

iii. The metrics-phase coupling process of choosing metrics corresponding to current software development phases.

The next chapter analyzes issues of software metrics use in the acquisition of software-intensive systems. The analysis addresses these issues from research on software metrics innovations and improvements done in the field of software engineering.

IV. ANALYSIS OF SOFTWARE METRICS IN THE ACQUISTION ENVIRONMENT

A. INTRODUCTION

This chapter analyzes the key issues regarding the use of software metrics by program managers in the management of software-intensive weapon systems. These are issues identified from interviews with program managers that have managed the acquisition of software-intensive systems as presented in Chapter III. The analysis is divided according to three themes:

i. The level of awareness and use of software metrics by PMs.

ii. Metrics used.

iii. The metrics-phase coupling process of choosing metrics corresponding to software development phases.

Innovations and improvements by researchers in the software engineering field are used to address these issues as there is a convergence between DoD objectives and advances made by these researchers. Estimation tools that facilitate the use of software metrics are also described as part of the analysis.

B. ANALYSIS OF THE KEY SOFTWARE METRIC ISSUES IN ACQUISITION OF SOFTWARE-INTENSIVE MAJOR WEAPON SYSTEMS

1. Level of Awareness and Use of Software Metrics by Program Managers

a. Software Management Training and Experience of Program Managers

All PMs were aware of software metrics and their purpose, however, none of the PMs who had received some training in software acquisition were trained in the development and use of software metrics in an acquisition program. Poor training in software acquisition appeared to contribute to PMs being overwhelmed by software

development details, which encouraged their delegation of software management to technical experts in the field. PMs usually designated personnel from the program office for management of software. In one case, software management was subcontracted to MITRE, a corporation that is a federally funded non-profit research contractor. This hands-off approach left the PM vulnerable to misrepresentation of software progress information by contractors.

Lack of training and the pressure of competing priorities also lead many PMs to adopt a management-by-exception approach to the software component. The use of metrics that can be graphically represented was favored by PMs because they are easy to understand. This saves the PM time to refocus on other priorities and relays important information to the PM for decision-making, which might otherwise leave the PM dependent on the recommendations of technical personnel who do not have the PMs' understanding of the overall system.

In May 2000, the Under Secretary of Defense (AT&L) Dr. Jacques Gansler emphasized the value of independent assessments performed for PMs [Ref 12]. This recommendation led to the sponsorship of the Tri-Service Assessment Initiative, which was instituted to provide an independent, objective review and analysis of software processes, product development, and integration. Numerous cost overruns, schedule slippages and performance issues motivated the focus on software. The goal of this initiative was to provide direct assistance to program managers as well as gain visibility over recurring issues that impact software-intensive systems acquisition. This initiative therefore offers PMs a chance to be less dependent on contractor's potentially subjective assessment. The initiative is also partnering with Defense Acquisition University (DAU) to insert future lessons learned into DoD-level acquisition education. To improve on the current training of PMs in the acquisition of system software, the initiative plans to use DAU's Executive Program Managers Course as a vehicle to identify problems and issues encountered in the acquisition community.

b. PMs Use of Metrics

Most PMs agreed that metrics offer high utility to software management, however, none said that they could rely on metrics alone to track progress. This is

justified by their wide experience and understanding of the dynamic and interdependent activities in major acquisition programs.

This view is the key issue discussed in a paper by N. Fenton and M. Neil [Ref 13]. The paper states that most software metric activities have not addressed their most important requirement: to provide information to support quantitative managerial decision-making during the software lifecycle. The authors state that it is for this reason, above all others that software metrics have failed to achieve a pivotal role within software engineering. One of the strategies proposed by the authors is the use of multiple criteria decision metrics. For example, from the interview data, it was found that number of defects is a commonly used metric during the EMD phase. The use of multiple criteria decision metrics is illustrated if along with defect measurement, the size of the program (using function points or SLOC) was measured, together with complexity metrics, which indicate the interdependencies of the CSU being tested. In this way, even if the number of defects was low but the complexity metric revealed many interfacing CSUs, the PM obtains better decision supporting aid that includes key factors largely missing from the usual metric data that measure isolated properties of software.

Another important issue is that metric development was hampered by PMs' lack of training in software metrics and is compounded by the necessity to comply with the requirements of the acquisition process. The interviews revealed that even though PMs only used metrics in the later stages of the acquisition, software metrics are required to be included in the statement of work, which occurs early in the acquisition cycle. This practice of software metric development is an indication of a lack of process maturity in DoD metric development practice. Table 2 below, shows the relationship of software measures to process maturity.

The current practice of metric development during early stages of acquisition management reflects a level one maturity. Development of metrics during these early stages appears to be ad hoc and chaotic as it is motivated by the need to satisfy acquisition process requirements rather than to serve as a risk reduction and decision support tool.

Maturity Level	Characteristics	Focus of Measurements
1. Initial	Ad hoc, chaotic	Establishing baselines for planning and estimating
2. Repeatable	Processes depend on individuals	Project tracking and control
3. Defined	Processes are defined and institutionalized	Definition and quantification of intermediate products and processes
4. Managed	Processes are measured	Definition, quantification, and control of sub processes and elements
5. Optimizing	Improvements are fed back to processes	Dynamic optimization and improvement across projects

Table 2: Relationship of Software Measures to Process Maturity. From Ref [15].

The actual use of metrics in the later stages, at best, reflects level two maturity. The length of acquisition programs almost ensure PM turnover, so that the metrics developed during the later stages of the program are according to the PM's experience with that phase of the program instead of being part of a long-term risk reduction strategy. Thus, the metrics process was developed according to the individual in the position of PM during the later stages, such as the EMD phase.

The lack of training in software metrics appears to contribute to the reason PMs underestimate the value of metrics during the early stages of the program. Thus, pressure to conform to acquisition policy requirements, coupled with a lack of understanding of the value of metrics in the early phases of the acquisition process, influences the PMs' selection and use of software metrics.

2. Analysis of Metric Effectiveness

All PMs interviewed were satisfied that the metrics commonly used to track software development were effective. Yet, they did not depend on metrics alone to support their managerial decision-making. In addition, metrics were used in the later

stages of the acquisition vice earlier stages where it was perceived that metrics would not be effective. This flawed practice was discussed in the previous section.

The most commonly used metrics were: SLOC, EVM and planned-versus-actual schedule to track cost and schedule; and resource utilization, manpower, requirements stability, requirements traceability, breadth of testing, fault profile and number of defects to track technical performance. This section analyzes the effectiveness of software metrics by comparing the effectiveness as experienced by PMs, with research findings on software metrics.

a. Hardware Metrics Versus Software Metrics: Earned Value Management and Schedule Charts

Most PMs used traditional hardware metrics to track cost and schedule, such as EVM to track costs, and schedule charts to monitor schedule. The problem with this approach is that the nature of software is different from that of hardware. For example, in the case of hardware, percentage completion is a suitable metric to track development progress. However, with software, implementing the last 10% of capability into a software system may require many times the amount of resources required for the completion of the previous 90% of the system.

b. Breadth of Testing, Fault Profile and Number of Defects Metric

PMs cited breadth of testing, fault profile and number of defects as test metrics commonly used during the EMD phase. These are further examples of metrics that are traditionally used for hardware systems. The use of these metrics in isolation is a poor indicator of quality [Ref 13]. The total correctness of proof for software is not economically feasible, and test plans for software systems perform partial correctness tests where test boundaries are set by operational profiles that the system has to perform. Thus, in the case of the number of defects metric, a high number of defects for a software system could be an indicator of the extensiveness of the tests rather than an indicator of poor quality. In fact, there is strong evidence that modules that are very fault-prone pre-release are likely to reveal very few faults post-release [Ref 14]. Conversely, the truly 'fault-prone' modules post-release are the ones that revealed no faults pre-release.

The latter point brings up an important issue in software engineering i.e., the lack of standards in this yet immature engineering field. This emphasizes the reliance on historical data, which can be extrapolated to give valuable estimates for current projects that are similar to past projects. These estimates are particularly valuable given the intangible nature of software. For example, if there were data on a similar past project, a better estimate of the technical performance of the current project could better be obtained.

The collection of data on historical projects is one of the recommendations by an SEI work group tasked to recommend measurement practices for DoD systems [Ref 15]. Their report recommends that DoD organizations define reporting formats and procedures for measurements taken on projects, with the ultimate goal of using data of completed projects, to support progress towards the management goals of current projects.

c. Source Lines Of Code

SLOC was a common metric used to track schedule and cost by the PMs interviewed. A disadvantage of using this metric is its application to programs written in languages that provide different levels of functionality. For example, a program written in Basic will require more lines of code than if C++ was used. In this case the higher number of lines of code gives the illusion of higher productivity when in fact the opposite is true. Newer programming languages, including Object-Oriented programming, cannot be compared to older ones through comparison of lines of code.

This shortfall may, however, be addressed if SLOC were measured according to a universally accepted set of rules. These rules should address how lines of code are counted, for example, how to handle comments compared to logical statements that contain functionality. It should also address all origins, stages of development, and forms of code production and distinguish between delivered and non-delivered statements, code that is integral to the product and external to the product, operative and inoperative (dead) code, master source code and various kinds of copies, and different source languages [Ref 15].

d. Manpower Metric

All PMs interviewed used the manpower metric to evaluate if the contractor has scheduled an adequate staffing level to produce the product within the time and budget allotted. The manpower metric is a principal means used by PMs to manage and control costs and schedule through planning and tracking human resources assigned to individual tasks and activities. This metric is one of the recommended metrics by an SEI work group tasked to recommend measurement practices for DoD systems [Ref 15]. They recommend staff-hours as a principal measure for the manpower metric. The staff-hour unit recommended follows the definition by the IEEE *Standard for Software Productivity Metrics*: "A staff-hour is an hour of time expended by a member of the staff".

Other candidates for measuring manpower data are labor-months and staff-week. The use of labor-months presents two obstacles: there is no standard for the number of hours in a labor-month and labor-months often do not provide the detail needed for measuring and tracking individual activities and processes, particularly when PMs' focus is on process improvement. Measuring effort in terms of staff-weeks presents many of the same problems and some additional ones as well. For example, although the basic assumption is that a calendar week is five working days, the length of a standard working day can vary between organizations. Weekend work, overtime work, and holidays falling within a week must also be addressed and defined if staff-week measures are to be used. By using staff-hours as the fundamental unit for recording and reporting effort data, these problems are avoided.

e. Requirements Traceability

All PMs interviewed considered this metric effective at its goal of ensuring that all requirements are implemented (to check for completeness of the system specifications) and that all CSUs implement system requirements (to identify unnecessary CSUs). This metric is just as suitable for hardware systems. Research by the Software Assurance Technology Center (SATC), an affiliate of NASA, recommend using the requirements traceability metric to track primary risks of the requirements phase [Ref

16]. The SATC has developed the Automated Requirement Measurement (ARM) tool to automate the data collection for this metric. The ARM tool is discussed in Section C.

3. Analysis of the Metrics to Phase Coupling Process

One of the areas of investigation for the interviews was the metric-to-phase coupling process used by PMs. The interviews revealed that the PMs did not adjust their metric set to the specific needs of different phases. Almost all metrics, except EVM and schedule charts, were used for the first time during the EMD phase. In addition, PMs did not see the value of using metrics early in the acquisition process, an issue previously discussed. This section explains the goal-question metric approach to developing software metrics to address these phase-specific issues. This is followed by the application of this approach by analyzing the changing software issues of different acquisition phases, then identifying software metrics suitable to address these issues, as researched from work done in the field of software management.

a. The Goal-Question Metric Development

One PM interviewed had metrics developed specifically for the program. The other PMs interviewed used metrics that were commonly used in other programs, without matching their management needs to metric development. A metrics program established without clear and specific goals and objectives is almost certainly doomed to fail [Ref 18]. Metric development requires a goal-related approach that ensures that the correct metrics are developed, and that the metrics being tracked, support some management goal. This is important given the significant amount of resources required for metric collection. During the first stage, managers must identify the metrics to collect. They do this by identifying primary goals of the overall program as well as phase specific goals. This is followed by identifying measures that will be used to determine progress towards these goals. This is the essence of Basili's Goal/Question/Metric (GQM) paradigm that provides guidelines that can be used to help identify measures to track software management goals [Ref 19].

b. Software Metrics for Specific Acquisition Phases

Software issues confronting the PM typically change by acquisition phase. However, the PMs interviewed did not account for this change in their choice of metrics during different phases of acquisition management. This section analyzes software metrics suitable for specific acquisition phases from research in software engineering, which could be used in the future by PMs. It analyzes the changing software issues of different acquisition phases, and then identifies software metrics suitable to address these issues, using research recommendations. Key software acquisition issues are listed by US DoD 5000.2 Software Life Cycle Phases [Ref 17].

Table 3 Requirements Phase. None of the PMs interviewed used metrics during this phase. They indicated that sufficient metric data could not be collected this early in the program. Their major program management issues of concern for this first phase of software development are requirements risk and resource implications.

Research from the field of software management indicated that metrics could contribute significantly to early acquisition phases. To address the first risk, the SATC [Ref 16] is working on methods to measure requirements document quality in much the same way that code quality is measured, that is, by measuring characteristics of the document itself. The metrics recommended to track document quality include the number of weak phases and options (as a measure of ambiguity), the number of To Be Determined (TBD) statements (as a measure of completeness), the percentage trace up and down (as a measure of requirements traceability) and the number of requirements changed (as a measure of volatility). The evaluation of all of the above metrics can be used to track requirements risk.

Estimates for function points, which can be obtained early in the development cycle are suitable metrics to estimate resources implications. At this stage of the acquisition process, the programming language may not yet be defined. Customer involvement is most important at this stage of the development. Function points and feature points do not depend on the source languages used. In addition, they are oriented

toward the customer's view (what the software does) rather than the producer's view (how he does it). These two properties make them suitable for use at this stage.

(2) Design Phase. None of the PMs interviewed used metrics during this phase. Once again, they indicated that sufficient metric data could not be collected this early in the program. The main program management issues of concern at this phase are reduction of technical, schedule and cost risks before coding begins.

Research indicates that metrics, such as the software complexity metrics, can be used by PMs to provide estimates valuable to the design of the software, which, is critical to all future stages of development. High complexity increases the propensity for errors and hence, increases risks to the schedule, budget and performance. The software complexity metric can, therefore, be used to track complexity of proposed designs and serve as a decision-support aid to PMs with the goal of reducing risk.

Structural complexity is measured using McCabe's cyclomatic complexity. It is the calculation of the number of test paths within a module. Coupling is measured by Fan-in and Fan-out. Fan-in is a count of the calls to a given module. This is the number of local flows into the module plus the number of data structures from which the procedure retrieves information. Fan-out is a count of calls from a given module. This is the number of local flows from a module plus the number of data structures, which the module updates [Ref 16].

(3) Code Phase. All PMs first began using metrics in this phase. They indicated that after the design phase, the system requirements stabilized and were fully understood and this enabled the use of metrics. The main program management issues of concern at this phase are that the software be maintainable and reusable. This facilitates the use of the requirements stability metric to track the amount of change made to the system, and the requirements traceability metrics to trace requirements to CSUs implementing system requirements. Both these metrics were used by the PMs interviewed.

The risks to schedule and reliability must be taken into account at this phase, but they are measured in the testing phase. Structure or architecture of

software, maintainability and reusability use the same metrics for evaluation, but with different emphasis and interpretation guidelines.

As discussed previously, no single metric is a precise risk determinant in the analysis of a product. The three metrics recommended for use are complexity, size, and the correlation of module complexity with size [Ref 16]. All PMs interviewed only used size metrics such as SLOC, without any correlation with other metrics. Complexity and size of modules are two common measurements of modules, but alone they provide a very one-dimensional picture. This was confirmed by the PMs interviewed who indicated that they did not feel confident relying on metrics alone. Software management indicates that a more comprehensive view of the data can be obtained by correlating the complexity to the size [Ref 16].

There are many different types of complexity measurements which include logical (cyclomatic) complexity, which is the number of linearly independent test paths; data complexity, which measures the data types and parameter passing; and calling complexity, which counts the number of calls to and from modules. Size may be measured by the SLOC metric, provided that a uniform and accepted set of counting rules is used.

(4) Testing Phase. All PMs interviewed used most software metrics, during this phase. The metrics they used, such as breadth of testing, number of defects and fault profile was used for performance testing. They indicated that at this phase, the code made the software product more tangible. Once code is generated and completed unit testing, formal testing begins that includes system integration and acceptance testing. During formal testing, all software modules are integrated cohesively and a series of system integration and validation tests are conducted.

All PMs indicated that the main focus at this phase is to find errors remaining from unit testing and errors resulting from unanticipated interactions between subsystems and components. PMs are also concerned with validating that the overall system provides functions specified in the requirements and that the dynamic characteristics of the system match those required. The goal of effective testing is to locate and repair faults in the software, to identify error-prone software, and to complete

testing on schedule with sufficient number of faults found and repaired that the software will operate as well as needed when it is put into operation. The main concerns in this phase are therefore schedule and reliability.

Software reliability is defined in statistical terms as the probability of failure free operation of a computer program in a specified environment for a specified time [Ref 16]. Software reliability is often measured by the "mean time between failure", but it can also be measured by estimating the number of errors left in the software that are of high criticality, that is, that prevents the software or some mission critical function from operating. This ensures a more cost effective approach to defect repair when the high cost of changing code at this stage is considered. The current practice by PMs interviewed is to measure number of defects without considering criticality.

During the testing phase, as the errors are identified, they may result in changes to the code. Research recommends the identification of code as having a high risk for errors that can now be linked to errors or change [Ref 16]. This linkage is important because changes to code, especially code that is high risk due to size and complexity, can lead to additional problems and ripple effect errors.

The need for this linkage also emphases the use of multiple decision criteria metrics. Size and complexity correlation metrics and comment percentage should be combined with the number of errors detected and the criticality of those errors. The comment percentage represents the amount of internal documentation that will provide information to the programmers making changes to the code. None of the PMs interviewed correlated their metrics. Minimal internal documentation can introduce new errors or ineffective changes. In this way, the use of metrics from the code and design phases provides the PM with a significantly better assessment of the software quality.

Correctness can be measured by stating the percentage of errors within a program that must be removed. The SATC is working to release the Waterman Error Trending Model for determining the status of testing by projecting the number of errors remaining in the software and the expected time to find a specified percentage of errors. The SATC is developing this model instead of relying on the Musa Reliability

model because it is less sensitive to data inaccuracy and provides for non-constant testing resources levels [Ref 16]. The limitations of the number of defects metrics, described earlier, is therefore reduced.

Analysis of completion rate data, which was not done by the PMs interviewed, can also be used by the PM to identify schedule risks at this stage. The completion rate of tests can be used to estimate the risk of completing all tests on time, hence completion of testing requirements and modules.

Finally, the location of faults that caused high criticality errors should be tracked. A concentration of them in a segment of code indicates a risk that the requirements are not well understood, or that the design is not suitable. All PMs interviewed did not attach criticality values to the defects identified.

C. AUTOMATED TOOLS FOR SOFTWARE METRICS

All PMs interviewed indicated that the use of software metrics demanded a significant portion of program resources, and metrics often competes unsuccessfully with competing priorities. Automating the collection and processing of software metrics not only reduces the amount of resources required, such as time and effort required to gather data, but also increases accuracy. This section describes some automated tools available for this purpose.

1. Automated Requirement Measurement (ARM) Tool

The Automated Requirement Measurement (ARM) Tool was developed by the Software Assurance Technology Center (SATC) at the NASA Goddard Space Flight Center as an early life cycle tool for assessing requirements that are specified in natural language. The objective of the ARM tool is to provide measures that can be used by project managers to assess the quality of a requirements specification document. The tool is not intended to evaluate the correctness of the specified requirements. It is an aid to "writing the requirements right", not "writing the right requirements".

The ARM tool searches each line of the requirements document for specific words and phrases the SATC has identified as quality indicators. Using these indicators, the ARM tool measures requirement completeness, ambiguity and other measures that indicate the quality of a requirements document.

2. COCOMO 81

COCOMO 81 is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity, according to software development practices that were commonly used in the 1970s through the 1980s. Listed by increasing fidelity, these forms are called Basic, Intermediate, and Detailed COCOMO. However, only the Intermediate form has been implemented by the University of Southern California in a calibrated software tool.

The implemented tool provides cost, effort, and schedule point estimates. It also allows a planner to easily perform "what if" scenario exploration, by demonstrating the effect adjusting requirements, resources, and staffing might have on predicted costs and schedules (e.g., for risk management or job bidding purposes). There are over 63 data points in the COCOMO 81 calibration database.

Dr. Barry Boehm originally published COCOMO 81 in 1981 under the simple name COCOMO. It is one of the most widely used software project cost estimation models.

3. CodeCount

The CodeCount toolset is a copyright of the University of Southern California's Center for Software Engineering. It is a collection of tools designed to automate the collection of source code sizing information. The CodeCount toolset spans multiple programming languages and utilizes one of two possible Source Lines of Code (SLOC) definitions, physical or logical.

The CodeCount toolset is provided in source code only, and may be used as is, modified or further distributed subject to certain limitations. The tools in the collection

are supplied in C source code only and the user is responsible for compiling and building executable.

The programs in the CodeCount toolset apply one of two possible SLOC definitions, physical or logical. The physical SLOC definition is based on Dr. Barry Boehm's Deliverable Source Instruction (DSI). It is programming language syntax independent, which enables it to collect other useful information such as comments, blank lines, and overall size, all independent of information content. The logical SLOC definitions will vary depending on the programming language due to language-specific syntax. The logical SLOC definitions are compatible with the SEI's Code Counting Standard.

4. PQMPlus

PQMPlus is a commercial product from Q/P Management Group. This tool uses the International Function Points Users Group (IFPUG) industry standard to estimate the size of a software system using function points. This package supports other project management activities such as the estimation of risk and estimate of resource implications based on the function points of the system. It also offers a range of work processor and spreadsheet type applications tailored to produce project management documentation such as schedule charts and project work plans.

D. CHAPTER SUMMARY

This chapter analyzed key software metric issues in acquisition of software-intensive major weapon systems. The first issue analyzed was the level of awareness and use of software metrics by PMs. This was followed by an analysis of metric effectiveness by comparing the effectiveness as experienced by PMs, with research findings on software metrics. Then, metrics suitable for specific phases were examined from research in the field of software management. Finally, some automated tools for the collection and processing of metrics were described. Metrics not currently used by PMs for the acquisition of software-intensive systems include requirement document characteristics,

function points, software complexity, number of high criticality errors, number of errors remaining, completion rate of tests and comment percentage. The next chapter presents conclusions and recommendations based on the findings of this analysis.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This section discusses the conclusions of the research on effective metric development and use for the acquisition of software-intensive major weapon systems. Table 3 shows the software metrics found to be most effective and the program management goals addressed by these metrics. Table 3 also shows the results of the metric to phase coupling analysis that sought to identify metrics that addressed the changing software concerns of the different phases during the software life cycle.

The results depicted in Table 3 were obtained from interviews with former PMs that managed the acquisition of software-intensive major weapon systems. Most PMs interviewed used software metrics, and indicated that they were effective, yet PMs did not rely on these metrics to give them a complete assessment of software progress. Moreover, all PMs interviewed had poor knowledge of proper software metric use and development. A lack of training in software management resulted in PMs delegating software management to software specialists. This management-by-exception approach adopted by PMs towards software management left the PM vulnerable to misrepresentation of program progress if the contractor was made solely responsible for reporting of software metrics. In addition, software specialists lack the PMs' program oversight, which is essential to choosing correct software metrics as a strategic risk-tracking tool.

Ideal metric development requires that the PM decide on the goals that he/she needs to meet to successfully manage the program. Metrics are then identified by their ability to track these goals. However, pressure to conform to acquisition policy requirements, coupled with a lack of understanding of the value of metrics in the early phases of the acquisition process, flaws PMs' use of software metrics.

Phase	Metric	PM Goal Addressed
All phases	Correlation between errors remaining, size and complexity metrics	Measures software quality.
Requirements	Requirement document characteristics (e.g. weak phrases, options, completeness)	Tracks requirements risk.
	Function points	Estimates resource implications.
Design	Software complexity	Estimates technical scope and identifies schedule risk.
Code	Requirements stability	Tracks changes in requirements.
	Requirements traceability	Traces back requirements to CSUs, and validates CSUs to requirements.
	SLOC	Estimates size of software system.
	Logical complexity	Measures reusability.
	Data complexity	Measures maintainability.
Testing	Number of high criticality errors	Measures system reliability
	Number of errors remaining	Measures correctness of a software system.
	Completion rate of tests	Measures schedule risk and risk of not completing testing requirements.
	Comment percentage	Facilitates proper code correction by communicating with programmers.

Table 3: Software metrics suitable for acquisition of software-intensive weapon systems.

A further problem with the current use of software metrics by PMs is the use of metrics in isolation, which gives a one-dimensional view. The acquisition environment is dynamic and complex. For software metrics to fulfill the true goal of decision support to

the PM, multiple metrics should be correlated and used to give a more complete view of software development.

This research concludes that it is not just the effectiveness of the software metrics used that were a problem, but rather their implementation and development that was flawed by the PM's lack of training in software management.

B. RECOMMENDATIONS

The poor training of PMs in software acquisition must be addressed. PMs are the key decision makers on acquisition programs and therefore need to have the educational background to make informed decisions. Defense educational institutions should train PMs on the importance of software metrics as a risk-tracking tool so that PMs would give their full commitment to its use and development.

Currently, the Tri-Service Assessment Initiative is underway to improve on the current training of PMs in the acquisition of system software and plans to use DAU's Executive Program Managers Course as a vehicle to identify problems and issues encountered in the acquisition community.

PMs need to start developing and using software metrics early in the acquisition process where it can offer valuable support to the crucial decisions made at this stage. The lack of training in software metrics is the main reason why PMs currently underestimate the value of software metrics during the early stages of the program.

The use of multiple criteria decision metrics is recommended to give the PM better decision support. Currently, most PMs use a single metric to track a specific risk or goal. However, no single metric is a precise risk determinant in the analysis of a product. While poor quality of any product is a risk to the specific goals of this project, some of the best measures of risk come form correlations of the base metrics. For example, complexity and size of modules are two common measurements of modules, but alone they provide a very one-dimensional picture. A more comprehensive view of the data can be obtained by correlating the complexity with software size.

The collection of project data is recommended for DoD □rganizations, which can be extrapolated to give valuable estimates for current projects that are similar to past projects. These estimates are particularly valuable given the intangible nature of software. This is due to the lack of standards in the as yet immature field of software engineering. DoD □rganizations should define reporting formats and procedures for measurements taken on projects, to facilitate ease of use of historical data with the ultimate goal of supporting management decisions by improving estimates of current projects.

PMs should be careful when using traditional hardware metrics to track cost and schedule, such as EVM to track costs and schedule charts to monitor schedule. The problem with this technique is that the nature of software is different from that of hardware. Therefore, the PM should adjust his/her expectations accordingly.

Lastly, the use of the goal-related approach to metric development is recommended. This approach ensures that the correct metrics are used, and that the metrics being tracked, support some management goal. This is important given the significant amount of resources required for metric collection.

C. ANSWERS TO RESEARCH QUESTIONS

1. What software metrics are suitable for the acquisition management of weapon systems software?

Table 3 shows the metrics suitable for the acquisition management of weapon systems software. These metrics are presented according to the phase that it should start being used so that it will address the changing software issues as development progresses. However, programs are unique and a goal-question approach to metric development should always be applied to ensure that the correct metrics are used.

2. What is the level of program managers' awareness of software metrics as a management tool for system software?

Program managers are aware of software metrics through DoD policies that mandated the use of software metrics for the acquisition of software systems. However, none of the PMs who had received some training in software acquisition, were trained in

the development and use of software metrics in an acquisition program. Poor training in software acquisition therefore caused PMs to be overwhelmed by software development details, which encouraged their delegation of software management to technical experts in the field. This hands-off approach left the PM vulnerable to misrepresentation of software progress information by contractors.

3. *What software metrics are currently used in the acquisition of major weapon systems and how effective are they?*

All PMs interviewed were satisfied that the metrics commonly used to track software development were effective. Yet, they did not depend on metrics alone to support their managerial decision-making. In addition, metrics were used in the later stages of the acquisition vice earlier stages where it was perceived that metrics could not be used.

The most commonly used metrics were SLOC, EVM and planned versus actual schedule to track cost and schedule; and resource utilization, manpower, requirements stability, requirements traceability, breadth of testing, fault profile and number of defects to track technical performance.

4. *What other software metrics could be used to improve system software management?*

Metrics not currently used by PMs for the acquisition of software-intensive systems include requirement document characteristics, function points, software complexity, number of high criticality errors, number of errors remaining, completion rate of tests and comment percentage. The use of multiple metrics by correlation was a recommended practice from software management research that was not currently practiced by PMs.

5. *What software metrics are appropriate for different phases of the acquisition process?*

Table 3 shows metrics that address the changing needs of software development phases.

6. *What estimation tools facilitate the use of these software metrics?*

The Automated Requirement Measurement (ARM) Tool, developed by the Software Assurance Technology Center (SATC) at the NASA Goddard Space Flight Center is an early life cycle tool for assessing requirements that are specified in natural language.

The CoCoMo 81 Intermediate software tool, implemented by the University of Southern California, provides cost, effort, and schedule point estimates. It also allows a planner to easily perform "what if" scenario exploration, by demonstrating the effect adjusting requirements, resources, and staffing might have on predicted costs and schedules (e.g., for risk management or job bidding purposes).

The CodeCount toolset is a copyright of the University of Southern California's Center for Software Engineering. It is a collection of tools designed to automate the collection of source code sizing information. The CodeCount toolset spans multiple programming languages and utilizes one of two possible Source Lines of Code (SLOC) definitions, physical or logical.

PQMPlus is a commercial product from Q/P Management Group. This tools uses the International Function Points Users Group (IFPUG) industry standard to estimate the size of a software system using function points. This package supports other project management activities such as the estimation of risk and estimate of resource implications based on the function points of the system.

D. RECOMMENDATIONS FOR FURTHER STUDY

1. Software Metrics in Automated Information Systems

The two major DoD software domains are Weapon System Software and Automated Information System (AIS) software. This research focused on Weapon System Software and not on AIS. While embedded systems interface with physical world entities, AIS systems interface with thousands of other AIS systems. Further research into AIS is therefore recommended to determine suitable metrics for AIS acquisition, which may differ from that for weapon systems.

2. Software Risk Management

This research proposed suitable metrics for acquisition management of software-intensive weapon systems. The implementation of these metrics as part of a risk management plan or a risk assessment model was excluded from the scope of this research and is recommended for further study.

3. Software Process Metrics

Project metrics, which are used to improve the project by tracking project attributes, was the subject of this research. Other metric categories include process and product metrics. Further study of process metrics, which are used to measure and improve the software development and maintenance processes, and product metrics, which are used to measure and improve the software product, is recommended.

4. Metrics for Object-Oriented Software

There is great interest in the use of the object-oriented approach to software engineering. This is due to a variety of claims about how it may improve the development of software, including factors such as greater reusability and increased extensibility. This research did not identify metrics suitable particularly for object-oriented software development. Research of metrics uniquely suited for object-oriented software development is therefore recommended.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. INTERVIEW DATA

(Target respondent: Program Managers of software intensive weapon system)

INTERVIEW 1: COL. DILLARD

1. *What programs were you involved in?*

i. Javelin missile – small shoulder launched anti-tank missile

ii. Army TACOMS missile system – big missile

iii. JASORS: Joint advanced special operations radio system- suite of low probability of detection and wide band high freq.

iv. All programs entered during the EMD phase

2. *Which domains of software did you work on?*

Command and control and embedded software. Metrics used in both domains. Prior to the edict on the use of metrics, we thought we knew where we were progressing and metrics were not used- focused on hardware. Did not have them to the extent that we had in 1993 forward. We seemed to know where the problems where but at the assistant PM level, even if we used metrics, I did not have the visibility of their use. Too wide an area to focus on, I typically had over 50 issues that we went over every week.

LEVEL OF AWARENESS

1. *What is your opinion of the role and usefulness of software metrics in management of system software?*

In 1993 the Army mandated the STEP metrics- 12 metrics- software test and evaluation metrics. I was not able to use all 12 in my program. The ones I used included cost manpower etc.

Metrics have a great utility in measuring progress, assessing cost, requirements tracebility. They are also good at providing a visual of something that is intangible and hard to inspect. Metrics make things less complicated. They allow a manager to be able to measure progress without having to know the intricacies of coding. Use of metrics does not guarantee any program slippages but metrics still give a lot of utility in management of a program.

2. Did you use software metrics in system software management?

If yes: Yes

2.1 How did you develop you set of metrics?

In my first two programs the metrics were mandated by the Army and I did not have to develop own metrics.

On another program, which was after 1993, about 4 metrics were written into contract. Either the government put into the statement of work or they were the common metrics used at that time. None of the metrics were developed by the PM.

The manpower metric was inserted in the JSOR program to get a handle on cost at subcontractor's location, which was agreed upon with the prime contractor. Use of this metric did not really involve additional cost or change of scope to gather data needed for the metric. This metric was not planned for, it was added after three years into the program.

2.2 Did you consider metrics process coupling i.e. choosing metrics corresponding to current software development phases?

No, I did not.

2.3 The goal/question/metric method is a popular approach in academic and industrial environments. Are you familiar with this method?

No, I am not.

2.4 What were metrics used for?

Metrics were used in the risk management plan, system engineering plan and the program management plan, and the integrated program summary which is a roll up of lots of documents, based on cost estimates, and acquisition program baselines. They were also used in the security classification test and evaluation master plan.

2.5 Was the use of software metrics mandatory? If it was not mandatory would you still use it?

Yes.

2.6 Were process improvements made as a result of the metrics? What follow up activities were performed to address shortfalls detected?

We had periodic PM reviews at the location of the contractor and would address metrics performance - done twice a year. On the missile program done 4 times a years. Actions stemmed out of the metric reports. On a day-by-day basis attention was given to problems that occurred. If metrics showed a problem, more effort placed on tracking the metric, did not wait for biannual review.

If no:

How did you manage system software?

N/A

3. Have you received any training in the management of system software?

ACRO 201 Acquisition level 1 2 and 3 in ACMO, and PM course in 1988. These courses don't take you very deep, they are not even as deep as software acquisition man 101 which any PM can benefit from having level 1 software acquisition. Even the level 3-acquisition requirement course, the block on software acquisition will not take PM to level 4. PM may read on articles but no detail of metrics given, it is just mentioned what they are.

4. Was a project software manager assigned explicitly for responsibility for software work products and activities?

Each PM office operated differently. In JASORS we had a GS 13, mid level man who was my software engineering point of contact. On most program offices like Javelin, I remember a software person on Army TACOMS.

Typically we would have one or two GS 13 person as software point of contact depending on the level of software development. More support was received from a nearby government lab RDEC (Research and development centers for the missile, aviation, tank automotive ,which is part of the PM office- where we buy support from. Each service has their commodity, a central laboratory for technology and a PMO for specific tech. The laboratory does applied research.

5. Was software management subcontracted?

Yes, laboratories were used. We also hired MITRE, a corporation, which is a federally funded research non-profit contractor. They rendered technical support to us.

6. Do you follow or research innovations and improvements in the field of software metrics

Yes. I have to know what's going on, I need to stay current. I read on-line publications e.g. DCMC.

7. If yes, do you feel these innovations/improvements could be used effectively by PMs in the DoD?

Yes. But I would still use what I used before. Need to make the contractor responsible for more of the design- previously government specified design. Now we ask the contractor which metrics he will be using. If he says none, we say at least testing and coding and progress metrics should be used in the trouble reports. We leave it to contractor who knows better how to design and adapt but there is inherent risk in this reliance on the contractors.

A. Metrics Used and Their Effectiveness

1. What metrics were generally used?

i. Manpower planned versus actual
ii. Computer resource utilization
iii. Software trouble reports: discovered versus resolved
iv. SLOC actual versus planned
v. Breadth of testing
vi. Number of defects

2. What metrics were used for tracking cost, schedule and performance?

Cost: Monthly cost performance report that was required by the Cost Accounting Standards report that we had in use at that time- today it is called the Earned Value Management. In that process the cost was broken down to the lowest cost account management level and an engineer is assigned as cost account manager for that piece of CSCI that has been functionally decomposed in the work breakdown structure. And so this cost performance report monthly was the cost metric, which was a roll-up by totals

and subtotals of major segments of the program and across 3 subcontractors and the prime contractor.

Schedule metrics: We had individual schedule charts, Gantt charts for each HW and SW CI as well as the master program schedule which showed milestones and inch-stones across time and where the slippage was- this was important in a program that was slipping as it is easy to draw a new Gantt chart to show the concurrency of activities that are going on, but the previous Gantt chart is not shown, and PM does not won't realize what the original inch-stone times were since the end date is the same. Therefore need some type of color-coding that can highlight slippage to show how much the program has moved over time- don't get the whole perspective.

Graphical representation of metrics is useful- it harder, in fact twice the information to show the slippage and today's baseline. But the old baseline is what's needed to show where you have given up ground. Then in technical performance there is lots of different metrics based on what your technical performance aspects of the system under development are. In Javelin it was focal plane array, sensitivity, detectivity, stability to seek, heat thermal image and that was portrayed along a technical performance growth curve and there was also others like uniformity, linearity, responsivity, and number of defects a number of technical perform parameters for a small piece in the front end of the missile. Weight, was a technical performance measure in the radio. Weight was a common technical metric in H/W and S/W. But software is different because you always have unknown errors after coding and even after testing there is no guarantees. At least on the hardware side it is easy to graph and see where your are going on the baseline you are trying to obtain.

3. What was the metric to objective match?

We did not develop the metrics.

3.1 Did the metric provide enough information to achieve the objective for which this metric was measured?

Yes.

4. Was metric data collected?

This was delegated to the software experts.

5. Were problems experienced in the collection of metric data?

Can't think of anything. Only asked for metrics that made sense. Everyone needed the metrics we asked for to know for themselves were doing. Did not want to impose any metrics that were over the top. Thus there were no problems.

6. Was there any problem gathering the information needed for the metric?

There was a variability problem with weight that is peculiar to any system. As components come in under invention they are designed but not built yet. They have a specification to be built to but you don't know on which end they will come in on. So the stack-up of all components can bust your tolerance if everyone comes in at the high end, so you only know how much you weigh at that final day-until product configuration is locked in.

7. How effective were these metrics in the management of system software development?

They were effective, in reinforcing what we already suspected and in confirming where we were having trouble with CSCI. This development spent between three major subcontractors of which one was really struggling and the metric highlighted the amount of dollars this contractor was spending and the amount of time overrun (8 weeks overscheduled).

Computer resource utilization was effective because we were approaching the top end of what our processing capability was going to be on the board. Thus the CRU metric was good to help us know that we were bumping up against the upper margin- we had a 30% margin in the spec that we were supposed to maintain over and above what the component item was going to endure.

The manpower metric– gathered from personnel used planned versus actual. In one subcontractor it was a real cost driver.

The errors discovered versus resolved was useful- showed what had to be done before Full-Up integration testing.

All its served usefulness. That is all the metrics I felt was needed and was against the mandate in 1993 compelling the use of all 12 metrics. Problem with the edict was that these metrics were imposed without the required resources. I tailored the metrics just as we are supposed to do today.

The mil-std 2167A was the standard at that time and it was a good management guide that was tailored to suit the program which saved the taxpayer money. So I tailored the use of the metrics to what I needed.

7.1 Where all three subcontractors asked to report the same metrics?

Yes. They did report the same metrics and they were easily comparable.

B. Metrics for Specific Phases

1. Where different metrics used for certain phases? If so, what were they?

All used during EMD. There was not any that I remember being used in the earlier phases.

2. How was phase specific metrics coupled with different development phases?

Software metrics don't mean much at earlier stages because system requirements are not fully understood. E.g. problem with SLOC is that program completion % based on original SLOC estimate is false because of requirements creep.

All programs worked on was in the EMD phase so I have a narrow perspective even though EMD is a very long and critical phase. I was not around when the guessitmating was going on. I was around were the coding testing was going on. So I was not around to see how effective the metrics was but I don't think metrics would be that useful. In the beginning metrics aren't useful.

3. At which phase is the metrics developed?

Prior to the letting of each contract metrics should be imposed on contractor. There are typically 3 or 4 contracts in the life cycle, and the first phase is studied for conceptual alternatives. And a prototype is likely to be a distinct contract for deliverable hardware that gets demonstrated by the end of a contract. And there may be a follow-on for phase 2 of the contract for advanced development so that don't have to go through RFP and contract award. But we had to include metrics in the statement of work prior to the prototyping phase and prior to advanced development. This is maybe why we have the edict, which is fine. But when contracts are already awarded the edict shouldn't apply.

New programs were required to comply. Metrics had great utility cannot manage software without it.

INTERVIEW 2: COL. DAVE MATTHEWS

1. What systems did you work on?

Multi launch rocket system

2. What phases of the programs were you involved in?

Field support phase and the R&D phase

3. Which domains of software have you worked on?

Embedded (sub-munitions side) and Command and Control.

I felt that embedded software was more challenging because you are frequently using application specific circuits (ASICs), which is firmware, to get it small enough to fit into hardware. Presents significant schedule risk because it takes a lot of work to reduce size."

A. Level of Awareness

1. What is your opinion of the role and usefulness of software metrics in management of system software?

They are extremely useful if it is the *right* metric.

2. Did you use software metrics in system software management?

If yes: Yes. The problem with mandatory is that you end up with the contractor converting his metric to what you need for a fee.

2.1 How did you develop you set of metrics?

The contractor developed it.

2.2 Did you consider metrics process coupling i.e. choosing metrics corresponding to current software development phases?

I never used this coupling process.

2.3 The goal/question/metric method is a popular approach in academic and industrial environments. Are you familiar with this method?

No.

2.4 What were metrics used for?

Risk management for me. For others it was used for cost management and estimating, status of software development by color-coding. We used less sophisticated metrics.

2.5 Was the use of software metrics mandatory? If it was not mandatory would you still use it?

Yes.

2.6 Were process improvements made as a result of the metrics?

Yes.

3. Have you received any training in the management of system software?

None. When to DCMC in 1985 and they did not teach software.

4. Was a project software manager assigned explicitly for responsibility for software work products and activities?

No we were totally dependent on the contractor.

5. Was software management subcontracted?

In some projects.

6. Do you follow or research innovations and improvements in the field of software metrics? If yes, do you feel these innovations/improvements could be used effectively by PMs in the DoD?

No, I do not.

B. Metrics Used and Their Effectiveness

1. What metrics were generally used?

Requirements traceability was used. Specification tree was also used for tracking. The most effective metric is integration testing. Easy to have software execute but interface specifications have to be sorted out and debugged during the integration testing. Complexity metrics very important here.

2. What metrics were used for tracking cost, schedule and performance?

As a level 06 PM I relied on my staff to work out the metrics – I managed more by exception. I was not involved in the metrics. Response to metrics done on priority of the metric based on what was the most cost effective use of the metric.

3. What were the metrics to objective match?

I do not what this was.

4. Did the metric provide enough information to achieve the objective for which this metric was measured?

Yes, however we never felt comfortable relying completely on the metrics to make our decisions.

5. Was metric data collected?

Yes, during testing.

6. Were problems experienced in the collection of metric data?

Yes, requirements evolve and then metric data is useless.

7. How effective were these metrics in the management of system software development?

Integration testing is important but there is still no guarantee that the system will perform error free. Contractors' metric was not good for me. No useful data for me as a manager. They have a great tendency to know how to build the watch but not what the time is.

C. Metrics for Specific Phases

1. Where different metrics used for certain phases? If so, what were they?

"No. We only used metrics during testing."

2. How was phase specific metrics coupled with different development phases?

"On fielded systems, software trouble report, quality deficiency reports are the metrics. Also used user conference during which feedback was the metric gained from troops that use the systems. From these we would prioritize what actions to take.

The government never has the expertise in house and is totally dependent on the contractor. How then does the government oversee the contractor development? Sure you have the SEI and but in a sole source contract negotiation the PM does not have confidence in the software until testing and even then there are always problems."

INTERVIEW 3: COL. TUDOR

1. What programs did you work on?

I worked on Army air defense programs- missile and guns. Air defense anti tank program- program was stopped but not because software was an issue failed for RAM issues.

Ten-twelve years ago another program I worked on was the called Army's Avenger system, which runs with the light division, sometimes with heavy divisions. The only time software became an issue on this program was when we had a to link to a command and control system. But here too, the system itself was a problem, the software was not a problem. Here the command and control system people took care of the software problems. We were never behind cost or schedule because they were more mechanical systems e.g. the missile was guided with a laser. Then air defense was done with the use of radar mostly.

2. Which phases?

On the missile and laser guided programs I worked during the EMD phases. On Stinger, Avenger- the IOT&E phase. Coding was never a problem on these programs. Integration which other parts of a system or subsystems was the problem.

I worked in the design phases early in my career. No cost or schedule increase, early in program, no problems experienced because too early yet. Most program managers are focused on cost, schedule and performance and that is what you are graded on.

A. Level of Awareness

1. What is your opinion of the role and usefulness of software metrics in management of system software?

I never used it.

2. Did you use software metrics in system software management?

If yes: No

2.1 How did you develop you set of metrics?

N/A

2.2 Did you consider metrics process coupling i.e. choosing metrics corresponding to current software development phases?

N/A

2.3 The goal/question/metric method is a popular approach in academic and industrial environments. Are you familiar with this method?

N/A

2.4 What were metrics used for?

N/A

2.5 Was the use of software metrics mandatory? If it was not mandatory would you still use it?

N/A

2.6 Were process improvements made as a result of the metrics?

If no: How did you manage system software?

N/A

3. Have you received any training in the management of system software?

Never. I am admittedly weak in that area. I went to program management course but software management was never taught.

4. Was a project software manager assigned explicitly for responsibility for software work products and activities?

Not that I can remember of. In the MEADS program it was still early on- around 1999 contract supposed to be awarded in 2000, positions were going to be developed later on.

5. Was software management subcontracted?

It was left up to the contractor.

6. Do you follow or research innovations and improvements in the field of software metrics? If yes, do you feel these innovations/improvements could be used effectively by PMs in the DoD?

No.

B. Metrics Used and Their Effectiveness

B. What metrics were generally used?

Never used metrics. Don't know if contractor used metrics.

C. What metrics were used for tracking cost, schedule and performance?

Cost and schedule never slipped so we never used them.

D. What were the metrics to objective match?

N/A

E. Did the metric provide enough information to achieve the objective for which this metric was measured?

N/A

F. Was metric data collected?

N/A

G. Were problems experienced in the collection of metric data?

N/A

H. How effective were these metrics in the management of system software development?

N/A

C. Metrics for specific phases

1. Where different metrics used for certain phases? If so, what were they?

N/A

2. How was phase specific metrics coupled with different development phases? What software intensive programs did you work on?

N/A

For lots of my programs I left software to integrators or contractors. I am talking about a 1990 time frame. On these programs, software took care of themselves. That's the way we viewed it- it was not issue on our programs, even though I did have knowledge that software was an issue on other programs.

I agree that there is a need for a better training of PMs since software is a major risk factor that affects cost, schedule and performance.

INTERVIEW 4: ART SANTA DONATO

1. Which program did you work on?

The AFATDS program.

2. Which phase of the program were you PM?

EMD. Because this phase is never done. There are always new versions to add more functionality- part of build a little test a little plan. There are always three versions: one being built, one being used, and one being planned to be build.

3. Which domain of software did you manage?

Embedded and command and control system.

A. Level of Awareness

1. What is your opinion of the role and usefulness of software metrics in management of system software?

You have to have metrics. Software is too complicated- without good metrics you will not know where you are in the process.

2. Did you use software metrics in system software management?

If yes: Yes

2.1 How did you develop you set of metrics?

I never developed the metrics. They were developed by the contractor's technical staff.

2.2 Did you consider metrics process coupling i.e. choosing metrics corresponding to current software development phases?

No.

2.3 The goal/question/metric method is a popular approach in academic and industrial environments. Are you familiar with this method?

No.

2.4 What were metrics used for?

They were not really used in the risk management plan but rather they were used as a risk mitigator. They were used in testing.

2.5 Was the use of software metrics mandatory? If it was not mandatory would you still use it?

Yes.

2.6 Were process improvements made as a result of the metrics?

Yes. In retests after failures, metrics brought attention to the way the contractor improved their testing and there was a significant reduction in failures.

But if the retest failed a second time, we sometimes found that the Kr just retested without making any changes. So now with the use of metrics we were able to identify the need for a process improvement and now if it goes to retest, it passes 90% of the time.

2.7 Is more useful to have a process improvement than a product improvement?

Yes definitely. SLOC was the first software metric that was a product metric but more code did not imply a better product. This is where process metrics are more valuable than product metrics.

3. Have you received any training in the management of system software?

No, I did not.

4. Was a project software manager assigned explicitly for responsibility for software work products and activities?

Yes, sometimes. But we most times we relied on the contractor.

5. Was software management subcontracted?

No. No need to as contractor had many software engineers. The Command and Control Electronics Command at New Jersey is the software engineering center for all maintenance of command and control support for the Army.

6. Do you follow or research innovations and improvements in the field of software metrics? If yes, do you feel these innovations/improvements could be used effectively by PMs in the DoD?

No. But my staff who are responsible for software reporting, do. I have a team of three people who manage the software metrics program.

B. Metrics Used and Their Effectiveness

1. What metrics were generally used?

Monthly reports from the contractor some weekly. Usually when getting close to release of software get a lot weekly metrics, regression testing and retesting and its good because you get a good feeling for how quick the contractor can fix the problems they have and I always like to look at the metric when they test something.

2. What metrics were used for tracking cost, schedule and performance?

The monthly metrics used:

SLOC, Cost (EVM,expenditure tracking), Schedule (chart), requirements stability, requirement traceability, breadth of testing, test progress, software build progress, fault profiles metric, computer resource utilization, software development progress. The cost and schedule metric was not software metric, they would be used on hardware as well.

3. What was the metric to objective match?

I am unable to answer that question, as I delegated the software metrics to others.

3.1 Did the metric provide enough information to achieve the objective for which this metric was measured?

No. You cannot just manage by software metrics. It is not the total answer to management.

4. Was metric data collected

Yes, It took 4 or 5 months before you got it the way you wanted but it was just a learning curve experience.

5. Were problems experienced in the collection of metric data?

Not really.

6. How effective were these metrics in the management of system software development?

They were very effective. Changes were made on an annual basis to the set of metrics used. If metric were found useless, they were omitted, and other metrics added if found valuable.

C. Metrics for Specific Phases

1. Where different metrics used for certain phases? If so, what were they?

Yes. Different metrics were used during test and EMD. Earlier phases used metrics like requirement tracebility and later phases used test and defects. You have to use metrics like SLOC together with other metrics. Process capability important to account for due to functionality being added, so that system not overburdened and do not need to change hardware. Therefore you need to track (CRU) processing capacity as a metric. Requirement management was a problem because the user and PM would agree on requirement and Kr misinterprets the requirement. Therefore needed traceability metrics to check what they built. It is an honest misinterpretation because the Kr personnel do not have a military background.

2. How was phase specific metrics coupled with different development phases?

No.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Defense Science Board, *Report of the Task Force on Military Software.* September 1987.
2. Software Technology Support Center, *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, Version 3.0, Volume 1, Department of the Air Force, Hill AFB, Utah, May 2000.
3. United States General Accounting Office, GAO/HR-97-9, *High-Risk Series: Information Management and Technology, Letter Report,* by Hinchman, James F., *Washington*, DC, February 1, 1997.
4. Information Resources Management Policies and Issues Accounting and Information Management Division, United States General Accounting Office, GAO/T-AIMD-97-38, *Managing Technology: Best Practices Can Improve Performance and Produce Results,* by Hoenig, Christopher, Washington, DC, January 31, 1997.
5. United States General Accounting Office, GAO/IMTEC-90-23, *Meeting the Government's Technology Challenge: Results of GAO Symposium.* February 1990.
6. Boehm, B.W., "Software Engineering Economics", Prentice Hall, Englewood Cliffs, NJ, 1981.
7. DSMC APMC SAM Teaching Note 3.1, Adapted from *Guidelines for Successful Acquisition and Management of Software Intensive Systems,* Volume 1, Version 3.0, Department of the Air Force, Hill AFB, Utah, May 2000.
8. Conte, S.D., Dunsmore, H.E. and Shen, V.Y., *Software Engineering Metrics and Models*; Benjamin/Cummings Publishing Company; 1986.
9. Fenton, Norman E. and Pfleeger, Shari L., *Software Metrics: A Rigorous and Practical Approach*, 2nd Edition, International Thomson Computer Press, London, UK, 1997.

10. Albrecht, Allan J. and Gaffney, John E. Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, November 1983.
11. U.S. Army, " Software Metrics", [www.army.mil/swmetrics], May 1999.
12. Baldwin, Kristen and Dwinnell, Laura, "Help Identify and Manage Software and Program Risk", Crosstalk: The Journal of Defense Software Engineering, Vol. 13 No. 11, November 2000.
13. Norman, Fenton and Neil, Martin, "Software Metrics: Roadmap*",* Proceedings of the 22nd International Conference on The Future of Software Engineering 2000, Limerick Ireland, June 2000.
14. Fenton, Norman E. and Pfleeger Shari L., *Software Metrics: A Rigorous and Practical Approach,* 2nd Edition, International Thomson Computer Press, 1996.
15. Technical Report, CMU/SEI-92-TR-19, *Software Measurement for DoD Systems: Recommendations for Initial Core Measures*, Carleton, Anita D., et. al., September 1992.
16. Rosenberg, Linda H. and Hyatt, Lawerence E., "Software Metrics Program for Risk Assessment", 47th International Astronautical Congress and Exhibition, Beijing, China, October 1996.
17. DAU APMC SAM Teaching Note 4.1, Adapted from *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, Volume 1, Version 3.0, Department of the Air Force, Hill AFB, Utah, May 2000.
18. Hall, Tracy and Fenton, Norman E., "Implementing Effective Software Metrics Programs", IEEE Software, 14(2) November1997.
19. Basili, Victor and Rombach, H.D., "The TAME Project: Towards Improvement Oriented Software Environment", IEEE Transactions on Software Engineering, June 1988.

BIBLIOGRAPHY

Augustine, Thomas, Capt.; Schroeder, Charles, "An Effective Metrics Process Model", Crosstalk: The Journal of Defense Software Engineering, Vol. 12 No. 6, June 1999.

Boehm, B.W., *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981.

Boehm, W. Barry; DeMarco Tom, "Software Risk Management", IEEE Software, 1997.

Conte, S.D., Dunsmore, H.E., and Shen, V.Y., *Software Engineering Metrics and Models,* The Benjamin/Cummings Publishing Company Inc., Menlo Park, CA., 1986.

Dekkers, A. Carol, "Managing (the Size of) Your Projects – A Project Management Look at Function Points", Crosstalk: The Journal of Defense Software Engineering, Vol 12 No. 2, February 1999.

Dekkers, Carol, "It is the People Who Count in Measurement: The Truth about Measurement Myths", Crosstalk: The Journal of Defense Software Engineering, Vol. 12 No. 6, June 1999.

Dwinnell, Laura, "Help Identify and Manage Software and Program Risk", Crosstalk: The Journal of Defense Software Engineering, Vol. 13 No. 11, November 2000.

Grady, Robert B., *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, New Jersey, 1992

Hall, Elaine M., *Managing Risk: Methods for Software Development*, SEI Series in Software Engineering, Addison-Wesley Longman Inc., Massachusetts, 1998.

Kastning, Jerry; Herman Jeff, " The Software Insight Tool: A Tool and Methodology for Risk Mitigation and CIO Assessments", Crosstalk: The Journal of Defense Software Engineering, Vol. 12 No. 8, August 1999.

Lee, J. Hearn, "Applications of Operations Research and Management Information System Concepts to Management of Large Software Projects", Doctoral Thesis, Illinois Institute of Technology, 1983.

Lucero, Don Scott, "PSM: The Army-DoD Tool to Implement Issue-Driven Software Measurement", Crosstalk: The Journal of Defense Software Engineering, Vol. 12 No. 6, June 1999.

Moller, K.H.; Paulish, D.J., *Software Metrics: A practioner's guide to improved product development*, Chapman and Hall, London, 1993.
Neitzel Jr., C. August, "Managing Risk Management", Crosstalk: The Journal of Defense Software Engineering, Vol. 12 No. 7, July 1999.

Nogueira de Leon, Juan C., *A Formal Model for Risk Assessment in Software Projects*, Doctoral Thesis, Naval Postgraduate School, Monterey, CA, September 2000.

Paulk, C. Mark, "The Capability Maturity Model: Guidelines for Improving the Software Process", CMU, SEI Series in Software Engineering. Addison-Wesley, 1995.

Perlis, A.; Sayward F.; Shaw M., *Software Metrics: An Analysis and Evaluation*, MIT Press, Massachusetts, 1981.

Poulin, A. Louis, "Assessing Software Risk", Crosstalk: The Journal of Defense Software Engineering, Vol. 13 No. 12, December 2000.

Riefer, Donald J., *Software Management*, IEEE Computer Society Press, 4th edition, Los Alamitos, CA, 1993.

Romero, James S., *Software Metrics: A Case Analysis of the US Army Bradley Fighting Vehicle A3 Program,* Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1998.

Rosenberg, H. Linda; Gallo, Al; Hammer, Ted; Parolek, Frank, "Continuing Risk Management at NASA", Crosstalk: The Journal of Defense Software Engineering, Vol. 13 No. 2, February 2000.

Software Technology Support Center, *Guidelines for Successful Acquisition and Management of Software Intensive Systems,* Volume 1, Version 3.0, Department of the Air Force, Hill AFB, Utah, May 2000.

U.S. Army, " Software Metrics", [www.army.mil/swmetrics], May 1999.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   Dyer Road
   Monterey, CA 93943-5101

3. Prof. Luqi
   Naval Postgraduate School
   Code CS/Lq
   Dyer Road
   Monterey, California, 93943-5000

4. Prof. David V. Lamm
   Naval Postgraduate School
   Code GSBPP/Lt
   Dyer Road
   Monterey, California, 93943-5000

5. Prof. Valdis Berzins
   Naval Postgraduate School
   Code CS/Be
   Dyer Road
   Monterey, California, 93943-5000

6. Prof. Man-Tak Shing
Naval Postgraduate School
Code CS/Sh
Dyer Road
Monterey, California, 93943-5000

7. Prof Mark E. Nissen
Naval Postgraduate School
Code GSBPP/Ni
Dyer Road
Monterey, California, 93943-5000

8. Col (Ret) Brad Naegle
Naval Postgraduate School
Code GSBPP/Na
Dyer Road
Monterey, California, 93943-5000

9. Col John Dillard
Naval Postgraduate School
Code GSBPP/Di
Dyer Road
Monterey, California, 93943-5000

10. Rakhee H. Ramgolam
ARMSCOR
370 Nossob Street
Erasmuskloof Ext 4
0181
Pretoria
Gauteng
South Africa