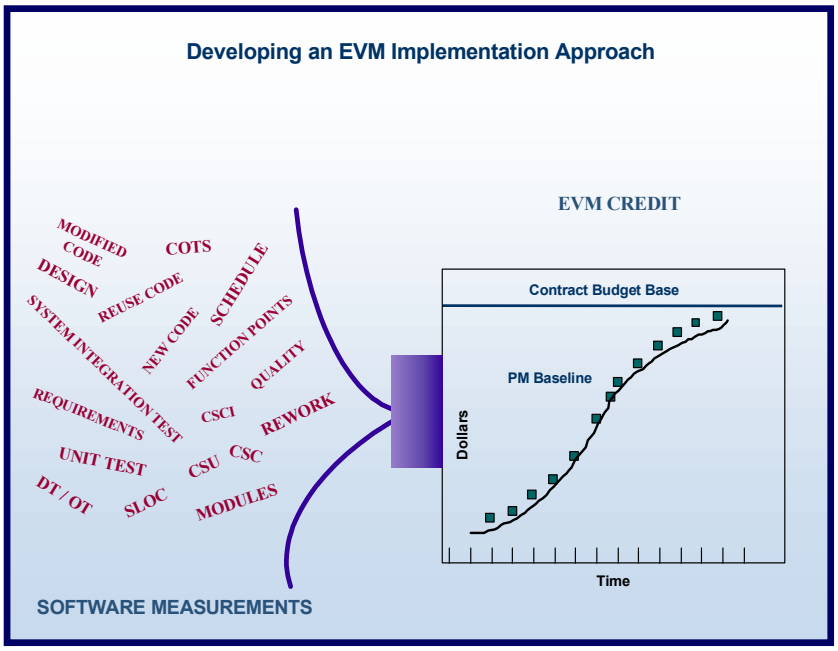
EARNED VALUE MANAGEMENT



Using Software Metrics
&
Measurements for Earned Value Toolkit

Dave Burgess
Cost Department Head

Ted Rogers
EVM Division Head

Chris Mushrush
EVM Subject Matter Expert

Dave Kester
EVM Subject Matter Expert

October 2004

Points of Contact
Process: Earned Value Management AIR 4.2.3
Technical: Software Engineering AIR 4.1.4

Intentionally left blank

RECORD OF CHANGE

Version	Date	Description of Revision
1.0	TBD	Initial release

FOREWORD

This handbook was created as a reference to provide guidance on how to develop an earned value approach to software development. Issues and challenges that must be considered and planned for and recommendations for software measures that can be used as the basis for determining earned value are presented.

The goal of the handbook is to provide software Program Managers, Technical Managers and Analysts with information needed to structure a robust earned value system that will provide essential information to aid in managing their program.

It is assumed that the reader is already familiar with the subject of earned value and its use. A copy of the Defense Systems Management College Earned Value Management Gold Card is provided in Appendix K for use as a reference for earned value concepts.

The handbook uses IEEE/EIA 12207 Software Lifecycle Process terminology to the maximum extent possible. Appendix I contains a comparison of MIL-STD-498 development activities to IEEE/EIA 12207.

Special Thanks

In addition to the many NAVAIR employees who assisted in the development of this handbook, we would also like to make special note of the assistance provided by Mr. Paul Solomon (Northrop Grumman Corporation and a Visiting Scientist at the Carnegie Mellon Software Engineering Institute), Mr. Mike Ferraro (Defense Contract Management Agency) and Mr. Robert Larrabee (ARINC). Without their assistance this handbook could not have been possible.

The handbook will be updated from lessons learned, best practices, and as continuous process improvements evolve at NAVAIR.

Authors: Rick Holcomb, Rick.Holcomb@navy.mil, (301)995-7657
Phyllis Sanders, Phyllis.Sanders@navy.mil (301)-342-0264

For copies or comments the following media are available:

By mail, contact:	Phyllis Sanders DEPARTMENT OF THE NAVY 4.2 COST DEPARTMENT 21491 GREAT MILLS ROAD LEXINGTON PARK, MD 20653 OFFICIAL BUSINESS	David Kester DEPARTMENT OF THE NAVY 4.2 COST DEPARTMENT 21491 GREAT MILLS ROAD LEXINGTON PARK, MD 20653 OFFICIAL BUSINESS
By telephone (voice), use:	(301)342-0264	(301)342-2403
By FAX (TELEX), use:	(301)342-2397	
By Email:	Phyllis.Sanders@navy.mil	David.Kester@navy.mil

TABLE OF CONTENTS

TABLES

FIGURES

1. Executive Summary

Earned Value Management (EVM) is a management tool that integrates cost, schedule, and technical performance. Despite its widespread use, EVM has often not been successfully implemented on software development efforts. There are several reasons why this is so:

1. Excessive use of Level of Effort (LOE).
2. Crediting full-earned value for tasks and requirements even though all tasks and requirements have not been completed.
3. Basing earned value on metrics and measures that do not directly relate to implementation of the software requirements.
4. Basing earned value on metrics and measures that are obsolete or inaccurate.
5. Utilizing EVM in isolation vice in conjunction with other software measurements and metrics to evaluate program status.
6. Failure to consider rework in developing the Performance Measurement Baseline (PMB).
7. Failure to correlate earned value with Technical Performance Measurement (TPM).

Developing an earned value implementation approach in software development is based on several software measures. Some of the more common software measures which can be used as the basis for earned value are summarized in Table 1-1.

LIFECYCLE PHASE	SOFTWARE MEASURE	BASIS OF EARNED VALUE	EARNED VALUE ALLOCATION METHOD	QUALITY AS AN EVM MEASURE
REQUIREMENTS ANALYSIS	System Requirements	Each System Requirement for S/W	0-100%	Good
	S/W Requirements	Completed effort for the S/W Requirement	0-100%	Good
	Function Point	Current Function Point Count	% Complete	Fair
	Schedule Milestones	ALL completed milestone tasks & requirements	0-100%	Poor
DESIGN	S/W Requirements	Completed effort for the S/W requirement	0-100%	Good
	Function Points	Current Function point Count	% Complete	Fair
	Modules	Each completed module	0-100%	Poor
CODE & UNIT TEST	S/W Requirements	Completed effort for the S/W requirement	0-100%	Good
	SLOC	Current SLOC estimate	% Complete	Poor
	ESLOC	Current ESLOC estimate	% Complete	Poor
	Function Points (FP)	Current FP count	% Complete	Fair
	Modules			
TEST	S/W Requirements	Completed effort for the S/W requirement	0-100%	Good
	Function Points	Count of successfully tested function points	% Complete	Fair
	Test Cases	Number of successfully completed SW test cases	0-100%	Good
	Modules	Each completed module.	0-100%	Poor
REWORK	S/W Requirements	Completed effort for the S/W requirement	0-100%	Good
	Function Points	Count of Function Points for completed rework	% Completed	Fair
	Software Defects	No recommended method.	N/A	Poor

Table 1-1: Common Software Measures Used as Basis for Earned Value

When selecting a measure upon which to base earned value, the best results are achieved when the measure is directly related to indicating that the desired functionality has been implemented. Requirements are directly related to implementing functionality and are the most effective measure for allocating earned value. Other measures less related to functionality result in reduced accuracy or additional effort and cost to implement.

Earned value is a formally defined measure and is only one of many measures that can be used to evaluate the status of a software project. Some measures are useful throughout the project lifecycle; some are applicable to specific tasks within specific development phases only. Each measure has its advantages and disadvantages. Like all measures EVM can provide objective information upon which to make project management decisions in order to achieve project functionality, cost and schedule goals. ***NO SINGLE MEASURE SHOULD EVER BE USED AS THE SOLE MEANS OF EVALUATING STATUS AND OF MAKING PROGRAM MANAGEMENT DECISIONS DURING THE SOFTWARE LIFECYCLE!***

Table 1-2 summarizes issues and challenges that must be considered and planned for when using a software measure to allocate the earned value for software development activities.

SOFTWARE MEASURE	EVM ISSUES	EXPLANATION
SLOC	Definition of SLOC	SLOC can be defined in many ways. There must be agreement on the counting methods and rules used to determine total lines of code.
	SLOC Counts	Estimated SLOC counts must be continuously updated as data becomes available throughout the development lifecycle.
	Effective Use As A Measure	SLOC as a measure is only appropriate for use during the Code & Unit test (C&UT) phase.
	SLOC Growth	Initial SLOC estimates are often low and should not be used. SLOC estimates and counts must be continuously updated as data becomes available throughout the development lifecycle.
	SLOC and Requirements	SLOC estimates and counts are based on planned functionality as defined by requirements. However, if all the planned functionality is not implemented, earned value based on the estimated SLOC will be overstated.
ESLOC	Similar to SLOC issues.	
FUNCTION POINTS	Trained FP counters	At least one team member must be a Certified Function Point Specialist (CFPS) to accurately account for the number of FPs for each task associated with the requirement.
	Software Requirements	FP counts are best performed on well defined software requirements specified at the level of detail found in a Software Requirements Specification (SRS).
	FP Counts	FP counts must be continually updated to reflect changes in requirements. While FPs can be applicable to all phases of software development, there may be specific tasks in each phase that are not well suited to earned value allocation.
MODULES	Module Completion As Sole Basis For EV	Using module completion as a measure by itself does not guarantee that all planned requirements for the module are implemented. Requirements must be tracked for each module to verify that all have been implemented as designed.
	Effective Use As A Meaure	Modules as a measure is only appropriate for use during the design, Code & Unit test (C&UT), and integration phases.
TEST PROCEDURES/CASES	Traceability to Requirements	Useful in determining if system and software requirements have been implemented correctly. Each test procedure or case must relate back to a requirement.
	Effective Use As A Measure	Test procedures and cases as a measure are only appropriate for use during testing phases.
MILESTONES	Milestone Completion As Sole Basis For EV	Using milestone completion as a measure by itself does not guarantee that all planned requirements for the module are implemented. Requirements must be tracked for each milestone to verify that all have been implemented as designed.
LEVEL OF EFFORT (LOE)	Effective Use As A Measure	LOE should not be used for any task that produces a product. LOE is more appropriate as a measure for indirect support activities such as management, administrative support, quality assurance, etc.

Table 1-2: EVM Issues For Software Measures Used as Basis for Earned Value

2. The Software Project

2.1 Introduction

Many Department of Defense (DoD) systems are extremely complex and software intensive. Each year costs for software design, development, implementation, and maintenance (i.e., upgrades, modifications, and fixes) continue to increase. Compared to other areas of engineering, software efforts experience large numbers of program terminations with large cost and schedule overruns.

In its 2000 Chaos Study, Standish Group International published the following data for a sample set of over 30,000 software programs:

1) 23% of software development projects were terminated,
2) 28% were successful,
3) 49% were challenged. On average the challenged programs experienced cost growth of 45%, schedule growth of 63%, and only 67% of the original requirements implemented.

Numerous articles, studies, and books have been written that address the various reasons why software programs fail. Capers Jones, in his book *Patterns of Software System Failure and Success*, lists several major factors associated with both success (i.e., on time with good quality) and failure (i.e., cancelled, delayed, or inoperable) of software projects. See Table 2-1, Critical Success Factors in Government Software Projects.

Projects that have better than average results in these critical factors have demonstrated that large complex software projects can be completed on time, within budget, and have few remaining defects after the software has been delivered.[1] But this requires cost and schedule performance to be monitored from the very beginning.

SUCCESSFUL PROJECTS	FAILING PROJECTS
Effective project planning	Inadequate project planning
Effective project cost estimating	Inadequate Effective project cost estimating
Effective project measurements	Inadequate Effective project measurements
Effective project milestone tracking	Inadequate Effective project milestone tracking
Effective project quality control	Inadequate Effective project quality control
Effective project change management	Ineffective project change management
Effective development processes	Ineffective development processes
Effective communications	Ineffective communications
Capable project managers	Inexperienced project managers
Capable technical personnel	Inexperienced technical personnel
Significant use of specialists	Generalists rather than specialists
Substantial volume of reusable material	Little or no reuse of technical material

Table 2-1: Critical Success Factors in Government Software Projects

The Software Program Managers Network (SPMN) developed the 16 Software Critical Practices™ for performance-based management. *Use Metrics to Manage* and *Track Earned Value* are two of the practices. [2] *Manage and Trace Requirements* and *Track Defects Against Quality Targets* are essentially specific types of software measures. The use of metrics, measures, and EVM, which is a rigorously defined type of metric/measure, is an essential combination to meeting another one of the critical practices: *Estimate Cost and Schedule Empirically.*

Measures and metrics are also required as part of achieving higher software development maturity levels under the Carnegie Melon Software Engineering Institute (SEI) Capability Maturity Model® for Software (SW-CMM®)[3] and the Capability Maturity Model Integration® (CMMI®)[4]. Appendix B contains information on the SEI capability maturity models.

2.2 Software Project Measurement

2.2.1 Software Metrics & Measurement References

Information on NAVAIR's software metrics policy and guidance on how to implement and interpret a robust flexible measurement program can be found in the following sources:

1. NAVAIR INSTRUCTION 5234.5 NAVAL AIR SYSTEMS COMMAND METRICS FOR SOFTWARE INTENSIVE PROGRAMS, 30 September 02, https://directives.navair.navy.mil/index.cfm
2. NAVAIR Software Metrics Program Handbook, SWDIV-HDBK-7, Rev 1, 1 November 2002
3. Practical Software Measurement, Objective Information for Decision Makers, John McGarry, David Card, Cheryl Jones, Beth Layman, Elizabeth Clark, Joseph Dean, Fred Hall, Addison Wesley 2002.
4. Practical Software & Systems Measurement, www.psmsc.com.
5. Practical Software & Systems Measurement: A Foundation for Objective Program Management, Version 4.0b, October 2000, http://www.psmsc.com/members/default.asp
6. Capability Maturity Model For Software (SW-CMM™), http://www.sei.cmu.edu
7. IISO/IEC 15939:2002, *Software Engineering - Software Measurement Process, (may be ordered at* http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=29572*)*

2.2.2 Base Measures, Derived Measures and Metrics

The Practical Software & Systems Measurement (PSM) methodology[5] defines both ***base*** and ***derived*** measures as they pertain to the PSM information model and measurement construct.

Base Measures are a measurement of a single attribute defined by a specified measurement method. Examples of a base measure are:

1. Estimate for the number of SLOC at a specific date for a specific module
2. Actual number of SLOC for a completed module
3. Number of open priority 1 defects on a specific date for a specific module of code.

Derived Measures are a measure or quantity that is defined as a function of two or more base and/or derived measures. Examples of derived measures are:

1. Productivity, the amount of code produced divided by the amount of staff hours it took to develop the code.
2. Requirements Change Rate, the number of software requirements at the beginning of a time period, divided by the number of new, modified or deleted requirements at the end of the time period.
3. Earned value, a base measure multiplied by its budgeted or planned cost.

Derived measures may use charts and graphs rather than mathematical functions to combine and/or compare base measures. Common examples are:

1. Time phased graphs of the change in the estimated size of the software over time and at various project milestones.
2. Time phased graphs of the numbers of open defects and their priorities over time and at program milestones.

Base measures can be thought of as raw data. On their own, they provide little information on the status of the project. A derived measure combines data from two or more base measures to provide insight into the actual status of the project and the basis upon which alternative corrective courses of action can be developed and program management decisions made.

Earned value is a formally defined derived measure. It is one of many measures that can be used to evaluate the status of a project during the software lifecycle. Some measures are useful throughout the project lifecycle; others are applicable to only specific tasks within specific development phases. In the majority of cases earned value is determined based on other derived software measures. The software measures discussed in section 3 provide a significant amount of useful information on project status in their own right. However, when used as part of an EVMS, they provide even greater insight into actual status of the program. No single measure should ever be used as the sole measure of evaluating status or to make program management decisions.

2.2.3 Using Measures As A Basis For An EVMS

Each program is responsible for determining what the drivers will be for its earned value system. Section 3 presents several different types of software measures that can be used to drive earned value during different development phases and tasks including their advantages and disadvantages. It could be argued that all of the measures are directly or indirectly required for organizations which have achieved a SW-CMM®[3] or CMMI®[4] level III certification. NAVAIR requires that Developers working on ACAT I, II, III and IV software intensive systems have achieved Level III[6] certification. Thus asking the developer to change the measure driving their earned value system should have only minimal or no impact on the cost of implementing their earned value and measurement program. In many cases, pointing out the potential weaknesses in the earned value implementation will be all that is required to convince the developer to modify

his implementation, especially if the alternative is an extensive discussion of how the weaknesses of the driving measure can be adjusted for and how it will not only impede the Government's insight into the program but also the Developer's.

Just because a developer's earned value system is compliant with EVMS, does not mean that the base and derived measures driving it will provide adequate information on program status[7]. Even if the developer is collecting better measures, there is still nothing in EVMS that requires the developer to use the better measures to drive their earned value program. Which means that the developer could insist on an earned value implementation with significant flaws in its ability to spot deviations from program cost, schedule, and technical objectives. **IN SUCH A SITUATION, IT IS ESSENTIAL THE GOVERNMENT HAS CONTRACTUALLY ESTABLISHED A MEASUREMENT PROGRAM THAT WILL INSURE THAT MEASURES CAPABLE OF IDENTIFYING DEVIATIONS FROM THE PROGRAMS COST SCHEDULE AND TECHNICAL OBJECTIVES ARE DELIVERED TO THE GOVERNMENT**. This provides the Program Office with an alternative method of spotting problems that does not rely on a potentially unwise earned value program. Problems that do arise for which the earned value system provided no warning provides additional justification for encouraging the Developer to modify their earned value to prevent other earned value failures in the future.

Even if the program does have an earned value system based on good measures, it is still important for program and technical management including project analysts to continue to analyze other software measures. No measure, no matter how carefully designed, can be guaranteed to spot every project problem. **OFTEN THE FIRST WARNING OF A PROBLEM WILL BE UNEXPLAINED INCONSISTENCIES BETWEEN DIFFERENT MEASURES.** Many times these inconsistencies at first glance indicate desirable trends but when taken together don't make sense.
If only earned value is being reviewed such inconsistencies will not be spotted and serious problems may go undetected until later in the program when the cost and delay associated with correcting them will be much more severe.

In addition to CMMI®, ISO/IEC 15939:2002 and PSM, additional guidance on implementing a robust flexible measurement program along with interpreting the results are available the in SW-CMM®[3], NAVAIR Instruction 5234.5[10], and the NAVAIR Software Metrics Program Handbook[8].

Often the best way of determining problems on a program is when one or more measures appear to provide contradictory information on the program status. This is usually due to an analysis that has not considered all the significant data or a misinterpretation of what a measure is really indicating. An analyst can be misled by a single measure. In order to get an accurate view of the actual program status, different measures must be viewed and analyzed together. An analyst looking at a group of measures and asking relevant questions about the inconsistencies and undesirable trends shown in the measures is much less likely to be misled and more likely to spot problems earlier in the effort when they can be more effectively dealt with. The effectiveness of earned value will be maximized when it is calculated based on the appropriate measure for the task and phase, and when it is evaluated in conjunction with other appropriate measures.

2.3 Project Management and Planning Activities

2.3.1 Peer Reviews

Software developers tend to be very optimistic about the progress they have made. More often than not the subjective judgment of the individual developer is used to determine how much of the work is finished. This often results in situations where +90% of a task is done very quickly, but the remainder of the task takes, weeks, months or even longer to complete.

Peer reviews are an alternative non-subjective means for determining when the work is actually completed. A peer review is a formal review by other members of the development team of the products produced by an individual developer for a specific development phase. This reduced subjectivity makes the successful completion of a peer review attractive as the indicator for when a task is actually completed and the earned value or BCWP for the task has been earned. Peer reviews are conducted as part of requirements analysis, design, code and unit test, and test procedure preparation.

If peer reviews are used as the basis for allocating BCWP, the tasks being reviewed must be broken down so that they can be completed in less than a month and preferably less than a week. This negates the need to use partial credit methods of allocating earned value and permits the exclusive use of the 0/100% method for earned value allocation. Longer task lengths do not allow enough opportunity for the developer to earn BCWP using a 0/100% method and force the use of other more subjective and less accurate methods.

The purpose of peer review is to detect and remove defects as early as possible from software products. The process involves a methodical examination of software work products by the producers' peers to identify defects and areas where changes are needed[9]. Peer reviews are included in the software CMM as a Level 3 key process area and cover all areas of the software life cycle as follows:

1. Requirements Analysis Phase. Verify that all higher-level requirements allocated to software in documents such as the MNS, ORD, Systems Specification, Software Performance Specification, etc. are decomposed into software requirements. Refer to the following for additional information:
   1. Executive Summary
   3.1.3.1 Software Requirements Analysis Phase (EVM & Requirements Metric)
   3.2.3.3.1 Software Requirements Analysis Phase (EVM & Function Points Metric)
   3.1.4 Deferred Functionality or Requirements
   3.1.5 Capacity & Performance Requirements Issues
   3.1.6 General Requirements Issues

2. Design Phase. Ensure that all software requirements assigned to the segment of the design under review have been implemented. Ensure that the design correctly implements the software requirements, defines a maintainable, extensible design, and implement appropriate corporate design methodologies. Refer to the following for additional information:
   1. Executive Summary

3.1.3.2 Software Design (Requirements Metric)
3.2.3.3.2 Software Design Phase (Function Points Metric)
3.3 Modules (EVM & Modules Metric)

3. Coding Phase. Ensure that all requirements and the design for the software module under review have been fully implemented and successfully unit tested. Ensure that the code is maintainable, that the code follows corporate coding standards. Refer to the following for additional information:
   1. Executive Summary
   2.7 Software Code Issues
   3.1.3.3 Code & Unit Test (C&UT) Phase (Requirements Metric)
   3.2.1 Code & Unit Test Phase (SLOC Metric)
   3.2.3.3.3 Code & Unit Test Phase (Function Points Metrics)

4. Test Phase. Verify that all software requirements can be traced to a test procedure and that the test procedure adequately exercises the code implementing all of the software requirements. Refer to the following for additional information:
   1. Executive Summary
   3.1.3.4 Test Phases (EVM & Requirements Metric)
   3.2.3.4 Test Phases (EVM & Function Points)
   3.4 Test Procedures/Cases (EVM & Test procedures/Cases Metric)

Numerous studies confirm that each peer review detects from 31 to 93 percent of defects, with a median of around 60 percent[10]. When the defects are caught much earlier in the development, they are less expensive to correct. Only 30% to 40% of defects are detected in each formal test conducted. Formal tests also detect errors much later in the development, when it is not only more costly to correct them, but when there is less time and resources to do so.

Peer reviews of software requirements, design, code and test procedures are required in organizations achieving SW-CMM® Level III certification[6]. NAVAIR Instruction 5234.1[8] requires all NAVAIR ACAT I, IA, II, III and IV software intensive systems to be certified at SW-CMM® Level III or its equivalent. Thus using successful peer review completion as the milestone for allocating earned value will not place an additional burden on the developer.

2.3.2 Integrated Baseline Review (IBR)

In addition to ensuring that the program has an executable schedule and budget profile, the IBR also provides the Government with an opportunity to evaluate how the developer plans to implement their earned value system including questions on any potential weaknesses in implementation. Specifically, what measures will be used by the developer for different tasks in different phases to determine how much earned value can be allocated and how the developer will integrate other measures into earned value in order to insure that the program status is accurately reflected.

To implement such a measurement system, the following steps should be done prior to the IBR:

1. Develop a well-structured program Work Breakdown Structure (WBS) that clearly differentiates software development tasks from hardware and systems engineering tasks in the program.

2. Identify Key Performance Parameters (KPPs) and Technical Performance Measures (TPM). See Appendix H for additional information.

3. Initiate a software risk management program and identify an initial set of program risks. If earned value is to be useful in identifying risk occurrence and the cost and schedule impact to the program, measures must be selected that provide insight into the identified projects risks.

4. Establish a Measurement IPT comprised of government and developer members. Most of the earned value concerns should be considered and addressed by the team. If a Measurement IPT has been established, then most of the earned value concerns cited in the IBR should have already been considered and addressed by the team.

Failure to do the above steps will reduce the ability to identify software related problems. Additional information on how to prepare and conduct an IBR can be found in the following documents:

1. The Program Managers' Guide to the Integrated Baseline Review Process, A Product of the Department of Defense / Industry Integrated Baseline Review Integrated Product Team, October 2002
2. NAVAIR Earned Value Management Integrated Baseline Review Toolkit, March 2003

2.3.3 Software Risk Management

EVM measures, like all measures, are intended to provide visibility into the status of the program in meeting its performance, cost, and schedule goals. A major aspect of meeting these goals is to ensure that the program's measurement program, including EVM measures, is able to alert program management of the likelihood of a risk occurring and the impact of the risk. The goal of the PSM[11,12,13] is to select measures which meet program information needs, the tracking of
program risks being one of the most critical. When deciding how to implement earned value on a program, the significant risks to the project must be considered when determining what measures will be used to drive the earned value system. ***EARNED VALUE MEASURES WILL PROVIDE NO INFORMATION ON THE STATUS OF A PROGRAM RISK IF THE MEASURES DRIVING THE EARNED VALUE SYSTEM GIVE NO INSIGHT INTO THE RISK.***

If your program has information needs or associated risks which none of the measures described section 3 give insight to, you will need to develop other measures and methods to drive the earned value system in order to achieve the necessary visibility.

It is also essential when selecting measures and setting up an earned value system that the benefit of tracking a specific risk or information need is determined for the program. A risk with a low

probability of occurrence or a low impact on the program if it does occur is unlikely to be worth the effort of collecting measures and calibrating the earned value system to track it.

2.3.4 Measurement IPT

The Measurement Integrated Process Team (IPT) is a government, developer team tasked with identifying program information needs and the measures that will provide insight into the program. The goal of the Measurement IPT is to maintain a flexible measurement program that accurately tracks the program status by discarding ineffective measures and replacing them with effective measures. Ideally, this team should be initiated prior to contract award in order to provide the competing developers with information on what the Government believes its information needs are for consideration in the developer's proposals. Once the winner is determined, they should immediately join the Measurement IPT to assist in determining what the contractual measurement requirements will be for the project.

The contract should be implemented so that the Measurement IPT is able to modify the program measures as the information needs of the program change over its life cycle. It is very unlikely that all of the program information needs and risks will be determined at the beginning of the program. Issues that appeared to be major programmatic risks initially may decrease in significance while others not originally foreseen will arise. This mandates that the measurement program have the flexibility to change with the developing information needs and risks.

In order to meet its objectives, each program is responsible for determining what the drivers will be for the earned value system. An EV system based on a robust flexible measurement program that adapts to the current program needs will be much more effective than one that is not. The Measurement IPT will play a critical part in ensuring that a program's earned value system is able to accurately track the program's progress in meeting cost, schedule and performance goals. An earned value system based on an inflexible measurement program is much less likely to provide this visibility, since its base measures will only identify a risk or problem by chance, rather than as a result of planning and forethought. CMMI®[4] (See Appendix B) and ISO/IEC 15939:2002[14] identify the requirements for implementing such a measurement program. PSM[6,7,8] provides a template and methodology for implementing the measurement requirements and goals.

2.4 Software Lifecycle Phases

A software lifecycle is the evolution of a software system from development through maintenance and eventually replacement. The duration of a lifecycle can be from days to years and can have as few as three phases or as many as 20 or more phases. Depending on the organization, their process, or their analytical tools, the phases of the software development process can differ in name and number. However, the same activities must be performed to complete the life cycle. The most common sequential activity phases and tasks performed under each activity phase are shown in Figure 2-1.

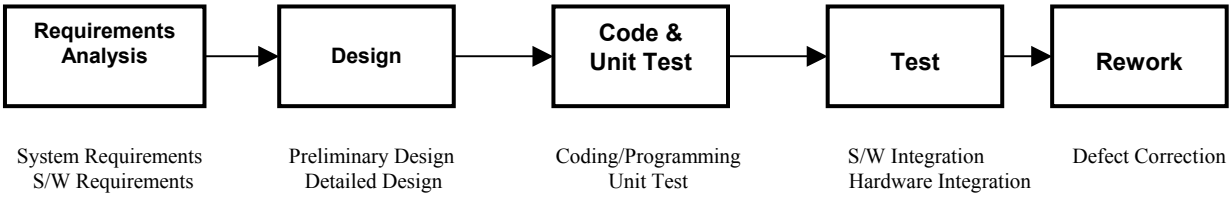


Figure 2-1: Software Life Cycle Activity Phases

The activity phase definitions below are based on Barry Boehm's book *Software Engineering Economics*[15] and *SEER-SEM™ User's Manual:* [16]

2.4.1 Requirements Analysis

Involves the creation of initial system requirements and related tasks and detailed software requirements analysis.

Refer to the following for additional information
1. Executive Summary

2.4.1.1 System Requirements

If the system has both software and hardware components, this normally is the time when specific functions are allocated to software. If it is possible to break out the software portion of this effort (identified by the work assignments, titles, and/or labor categories of the personnel who participate in the process), the software portion should be assigned to the specific software project under analysis.

2.4.1.2 Software Requirements

This activity uses information created during system requirements analysis. There should be no difficulty in identifying and assigning this effort to a specific software project. Since many software analysis models do not distinguish between system level and software level requirements effort, it is important to define terms and understand exactly what effort is included in the estimated and actual labor hours.

Refer to the following for additional information:
1. Executive Summary
3.1.3.1 Software Requirements Analysis Phase (EVM & Requirements Metric)
3.2.3.3.1 Software Requirements Analysis Phase (EVM & Function Points Metric)
3.1.4 Deferred Functionality or Requirements
3.1.5 Capacity & Performance Requirements Issues
3.1.6 General Requirements Issues

2.4.2 Design

Involves breaking the software into packages and/or functions. This effort may be done formally or informally. Some software analysis models combine preliminary and detailed design into one activity phase.

Refer to the following for additional information:

1. Executive Summary
3.1.3.2 Software Design (Requirements Metric)
3.2.3.3.2 Software Design Phase (Function Points Metric)
3.3 Modules (EVM & Modules Metric)

2.4.2.1 Preliminary Design

The data flows between different program components may be defined, and the design mapped back into the software requirements.

2.4.2.2 Detailed Design

Includes the further definition of software down to the single decision point.

2.4.3 Code & Unit Test (C&UT)

Includes writing the actual source code and testing it at the unit or function level. The programmer often performs unit testing as part of the coding process.

Refer to the following for additional information:

1. Executive Summary
2.7 Software Code Issues
3.1.3.3 Code & Unit Test (C&UT) Phase (Requirements Metric)
3.2.1 Code & Unit Test Phase (SLOC Metric)
3.2.3.3.3 Code & Unit Test Phase (Function Points Metrics)

2.4.4 Test

Includes testing of the software to determine if requirements are being met. Testing can be done formally or informally. Some software analysis models combine Code & Unit Test, Component Integration & Test, and Program Test into one "Programming" or "Coding" phase.

Refer to the following for additional information:

1. Executive Summary
3.1.3.4 Test Phases (EVM & Requirements Metric)
3.2.3.4 Test Phases (EVM & Function Points)
3.4 Test Procedures/Cases (EVM & Test procedures/Cases Metric)

2.4.4.1 Computer Software Component (CSC) Integration and Test

This is integration of software units with other units to form a computer program.

2.4.4.2 Computer Program (CSCI) Test

This is testing of each CSCI as a whole.

2.4.4.3 Software and Hardware System Integration & Test

Involves software to software and software to hardware integration. The final system is tested with live data in a real world type environment. Some software analysis models separate System Integration & Test into its two component parts: SW-to-SW integration and SW-to-HW integration.

2.4.5 Software Rework

Software rework is the correction of defects. These defects may be in the requirements, design and other documents, or in the code itself.

Refer to the following for additional information:
1. Executive Summary
3.1.3.5 Software Rework (EVM & Requirements Metric)
3.2.3.3.5 Rework (EVM & Function Points Metric)
3.5 Software Defects (EVM & Software Defects Metric)

Software maintenance when added to development defines the complete software lifecycle cost. Software maintenance is defined as the process of modifying existing operational software while leaving its primary functions intact. Software maintenance can be classified into three main categories:

1. Corrective, corrects processing, performance, or implementation failures
2. Adaptive, changes in the processing or data environment
3. Perfective, performance or maintainability enhancements

2.5 Software Development Models

DoDI 5000.2, Operation of the Defense Acquisition System, 12 May 2003, Section 3.3 provides the following definitions of Evolutionary, Spiral and Incremental development:

Evolutionary Acquisition is the preferred DoD strategy for rapid acquisition of mature technology for the user. An evolutionary approach delivers capability in increments, recognizing, up front, the need for future capability improvements. The objective is to balance needs and available capability with resources, and to put capability into the hands of the user quickly. The success of the strategy depends on consistent and continuous definition of requirements, and the maturation of technologies that lead to disciplined development and production of systems that provide increasing capability towards a material concept. The approaches to achieve evolutionary acquisition require collaboration between the user, tester, and developer. Each has its advantages and disadvantages. The development approach may be specified in a contractor's Software Development Plan (SDP).

2.5.1 Spiral Development Lifecycle Model

In this process, a desired capability is identified, but the end-state requirements are not known at program initiation. Those requirements are refined through demonstration and risk management; there is continuous user feedback; and each increment provides the user the best possible capability. The requirements for future increments depend on feedback from users and technology maturation. Figure 2-2 shows the IEEE Software Spiral Development Lifecycle Model.

2.5.2 Incremental Development Lifecycle Model

In this process, a desired capability is identified, an end-state requirement is known, and that requirement is met over time by developing several increments, each dependent on available mature technology. Figure 2-3 illustrates the incremental model based on Barry Boehm.

The waterfall model is only rarely used in NAVAIR software development and when used, it is for relatively simple efforts.

2.5.3 Waterfall Development Lifecycle Model

In the traditional model, each stage is a prerequisite to subsequent activities. Successful completion of a stage is required before starting the next one. Life-cycle reviews should be used to assess progress and determine whether or not to proceed to the next phase of software development. Following is a list of assumptions about the waterfall model:[17]

1. The requirements are known in advance of implementation.
2. The requirements have no unresolved, high-risk implications. (i.e. risks due to COTS choices, cost, schedule, performance, safety, security, user interfaces, organizational impacts.
3. The nature of the requirements will not change very much during development.
4. The requirements are compatible with all key requirements' stakeholders expectations.
5. The right architecture for implementing the requirements is well understood.
6. There is enough calendar time to proceed sequentially.

Figure 2-4 shows the waterfall model based on DoD-Std-2167A.

[1] Jones, T. Caper, "Government Software Projects Rank High in Major Critical Success Factors", Crosstalk, Vol.15
[2] Software Program Managers Network 16 Critical Software Practices, http://www.spmn.com/16CSP.html
[3] Key Practices for the Capability Maturity Model Version 1.1, CMU/SEI-93-TR-025, http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.025.html
[4] CMMI® Models, http://www.sei.cmu.edu/cmmi/models/
[5] Practical Software Measurement, Objective Information for Decision Makers, John McGarry, David Card, Cheryl Jones, Beth Layman, Elizabeth Clark, Joseph Dean, Fred Hall, Addison Wesley 2002. page 17 – 29.
[6] NAVAIR INSTRUCTION 5234.5 NAVAL AIR SYSTEMS COMMAND METRICS FOR SOFTWARE INTENSIVE PROGRAMS, 30 September 02, https://directives.navair.navy.mil/index.cfm
[7] Using CMMI to Improve Earned Value Management, Paul Solomon, October 2002, CMU/SEI-2002-TN-016, http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tn016.pdf
[8] NAVAIR Software Metrics Program Handbook, SWDIV-HDBK-7, Rev 1, 1 November 2002
[9] Capability Maturity Model For Software (SW-CMM) Key Process Area: Peer Reviews, http://www.sei.cmu.edu
[10] Basil, Victor R., Boehm, Barry, "Software Defect Reduction Top 10 List", Crosstalk, January 2001
[11] Practical Software Measurement, Objective Information for Decision Makers, John McGarry, David Card, Cheryl Jones, Beth Layman, Elizabeth Clark, Joseph Dean, Fred Hall, Addison Wesley 2002
[12] Practical Software & Systems Measurement, www.psmsc.com
[13] Practical Software & Systems Measurement: A Foundation for Objective Program Management, Version 4.0b, October 2000, http://www.psmsc.com/members/default.asp
[14] ISO/IEC 15939:2002, *Software Engineering - Software Measurement Process*
[15] "Software Engineering Economics", Barry W. Boehm, 1981, Prentiss Hall PTR
[16] "SEER-SEM™ Version 6.0 User's Manual, Galorath, Inc., 2001, printed in USA
[17] "Spiral Development: Experience, Principles and Refinements", Barry Boehm USC Spiral Development Workshop, 9 February 2000, http://www.sei.cmu.edu/cbs/spiral2000/Boehm/sld001.htm

Spiral Development Lifecycle Model

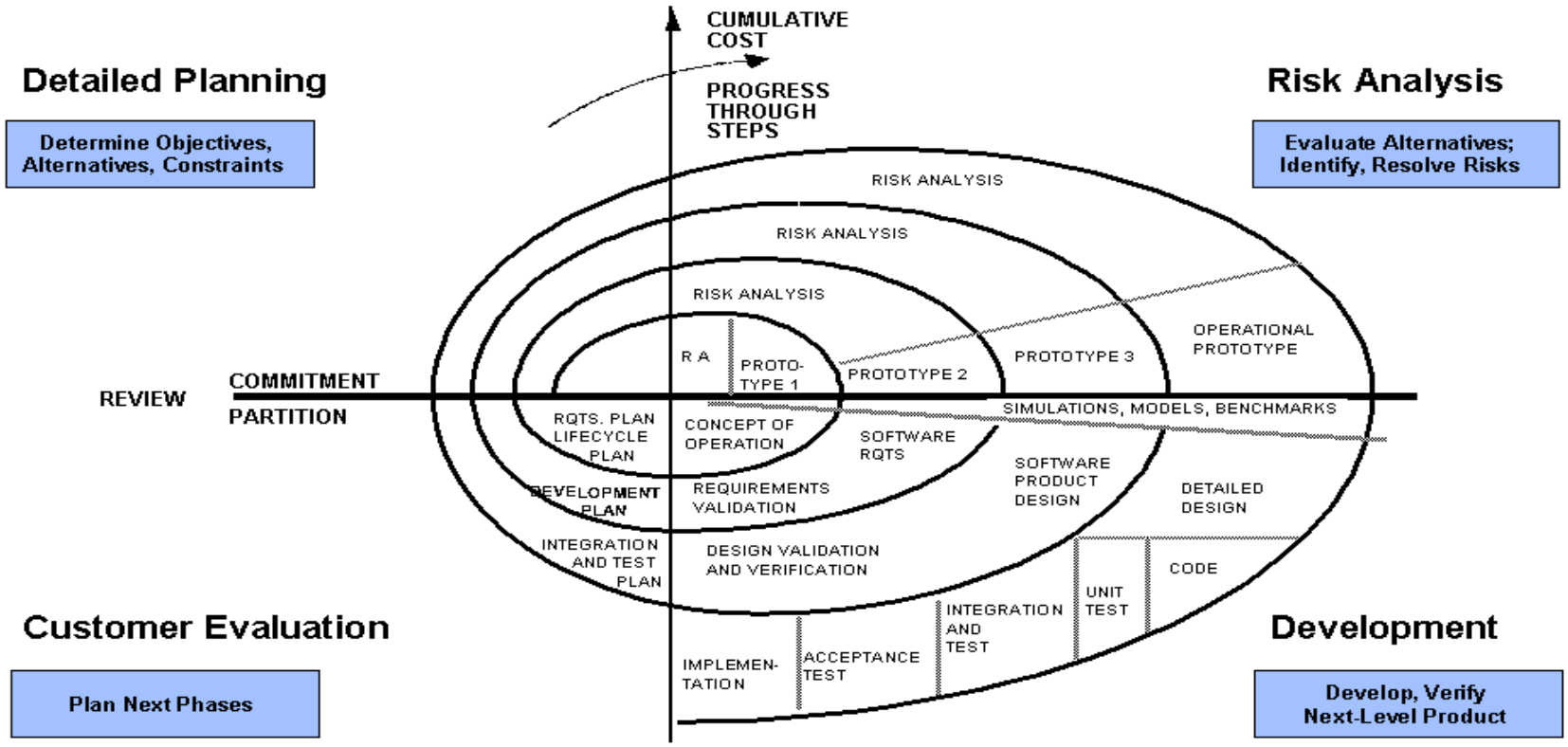


Figure 2-2: IEEE Software Spiral Development Lifecycle Model

Incremental Development Lifecycle Model

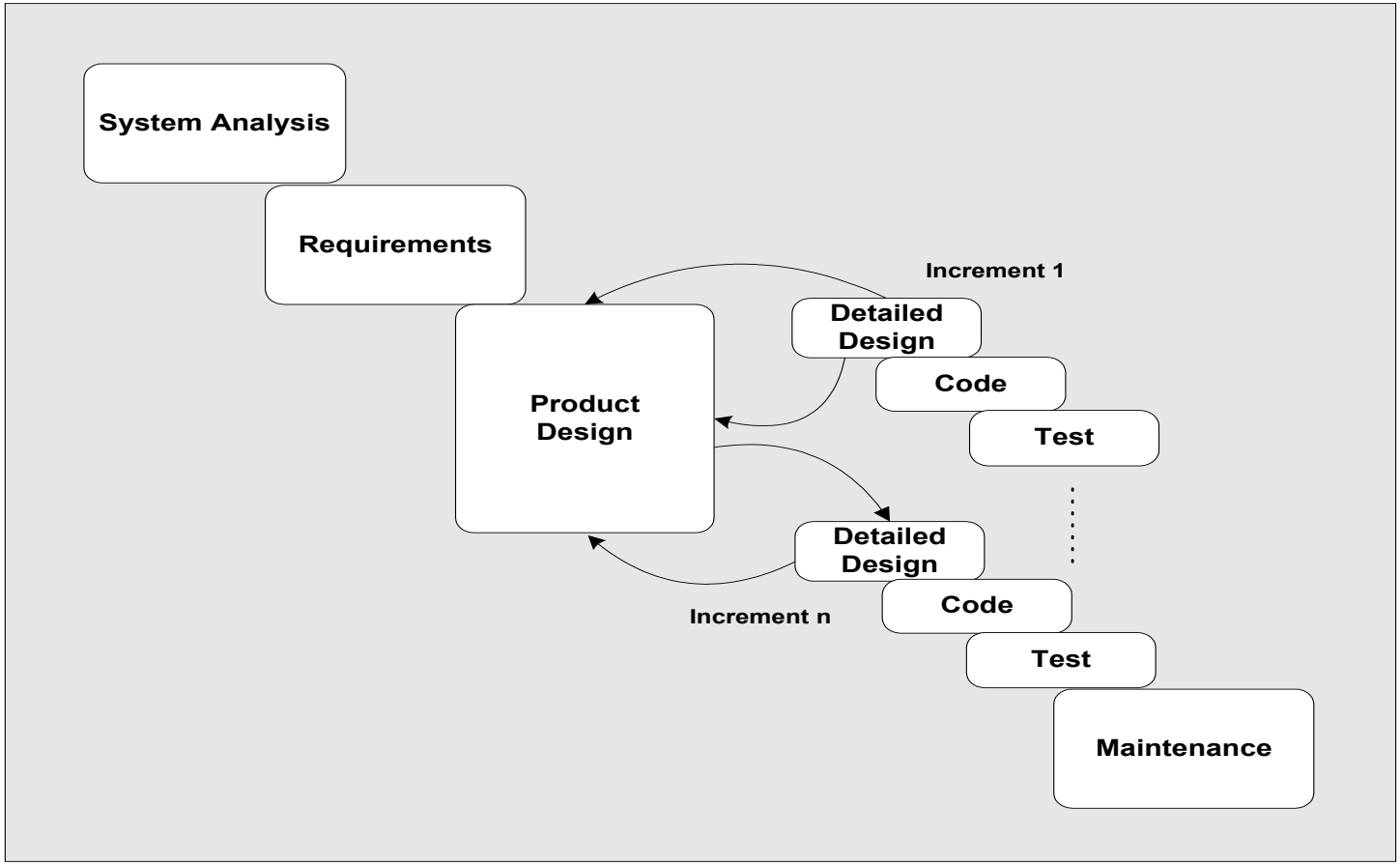


Figure 2-3: Software Incremental Development Lifecycle Model

Waterfall Development Lifecycle Model

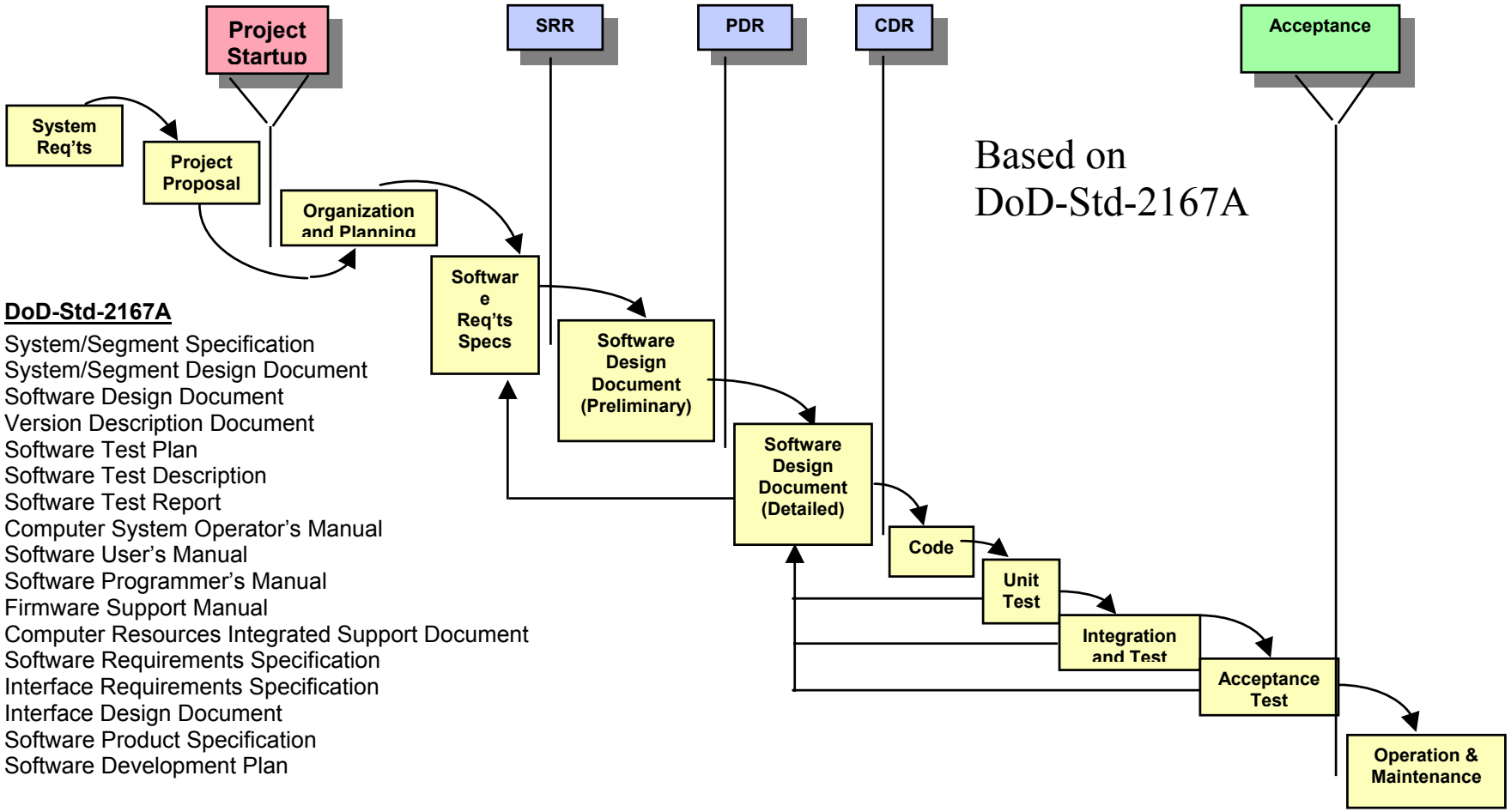
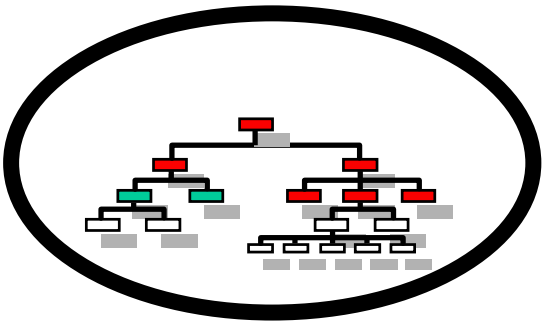


Figure 2-4: IEEE Software Waterfall Development Lifecycle Model

2.6 The Software Work Breakdown Structure (WBS)



A well-designed software WBS is essential to an efficient and accurate EVM approach. A WBS structure that does not differentiate clearly between software and hardware tasks or provide adequate visibility into the software tasks results in efforts that do not provide management with visibility into problems with cost and schedule.

While the guidance in DoD MIL-HDBK-881 emphasizes the importance of software within the DoD environment, some of the guidance is inconsistent with the need to develop a software WBS compatible with allocating earned value. Specific problem areas in MIL-HDBK-881 are:

1. Section 2.2.5 Avoiding Pitfalls In Constructing a Work Breakdown Structure. Appendix C contains Section 2.2.5 wording.

   Section 2.2.5 identifies phases as items that should not be included separately in a WBS. It states that the WBS should "address the products required, NOT the functions or costs associated with those products". While phases may not be appropriate for inclusion in a WBS, the various deliverables and/or artifacts produced during these phases are. The requirements phase produces the software requirements specifications. The design phase produces design documents. Code and unit test phases produce source code and verifies the functioning of the code according to test plans and procedures. Thus when you see phases identified in a software WBS it does not indicate noncompliance with the handbook, instead it is identifying the development task leading to a significant product deliverable or artifact. Failure to include these phases in the software WBS delays project visibility because specific status and progress on these deliverables is not shown.

   Due to the length of time involved with developing software, especially in larger efforts, a method of breaking the software development into tasks of a manageable schedule length is required to effectively monitor EVM. Breaking the development effort into phases or breaking the WBS into smaller modules of code results in tasks of shorter schedule length, thus contributing to EVMs accuracy and provide early warning of project problems.

   Dr. Barry Boehm, the primary developer of COCOMO (COnstructive COst MOdel) and COCOMO II software cost and schedule estimation models, includes the phases in software WBSs in his various works[1,2]. In addition, virtually all software cost and schedule estimating

software tools, including tools produced by Galorath (SEER™), SPR (KnowledgePlan), COCOMO, COCOMO II, PRICE Systems (PRICE-S) implicitly recommend the use of these phases by basing the estimates produced by their tools on these phases.

The software WBS should be customized to the program's software requirements. The WBS is necessary for identifying all tasks and issues, which have labor, schedule and material costs assigned to them as part of the complete estimate.

2. Section 3.2 Contractual Issues & 3.2.1 Software and Software Intensive Systems. Appendix D contains Sections 3.2 and 3.2.1 wording.

Experience has shown that software development has an inherently high technical, cost and schedule risk[3,4,5,6]. It is vital that the software WBS be designed to insure an effective EVM system can be implemented. It is essential that appropriate reporting be provided to at least the top level of each software development effort comprising the system.

For a large system, there may be several software development efforts associated with different hardware system components. MIL-HDBK-881 requires that software identified in the WBS be associated with the specific piece of equipment it will operate on in the system. This often results in software WBS elements not appearing until below the third level of the WBS. It such cases it must be contractually specified that EVM be reported for software elements, even when such elements are below the third level of the WBS. At a minimum EVM must be reported to the top level at which each software component appears in the WBS. It is preferable that EVM be reported to the level where the software development phases (requirements analysis, design, code & unit test, integration test, system test) appear in the WBS structure. The WBS can be tailored beyond the level prescribed in MIL-HDBK-881 to provide increased visibility into program performance, this is highly recommended due to cost and risk associated with software efforts. Figure 2-5 is an example of a WBS that provides visibility into software development efforts.

Contract WBS - SW Treated as subsystem
1(3) 2(4) 3(5) 4(6) 5(7)
Fire Control
Radar
Receiver
Applications Software
Build 1
Build 2
Integration Testing
Systems Software
Build 1
Build 2
Integration Testing
Transmitter
Antenna
Radar Integration
Platform Integration

Figure 2-5: Contract WBS Example

A software project summary WBS contains the most suitable WBS items that satisfy operational needs. The WBS elements are described generically and apply to each type of system. The associated activities and deliverables are listed with each software WBS description. The application software refers to all the effort required to design, develop, integrate, and checkout product applications, builds, and CSCIs. It does not include software integral to any specific hardware subsystem specification.

Figure 2-6 [7] illustrates an excellent template for a multi release, incremental or spiral development software project WBS. It shows the breakdown of both application and system software CSCIs.

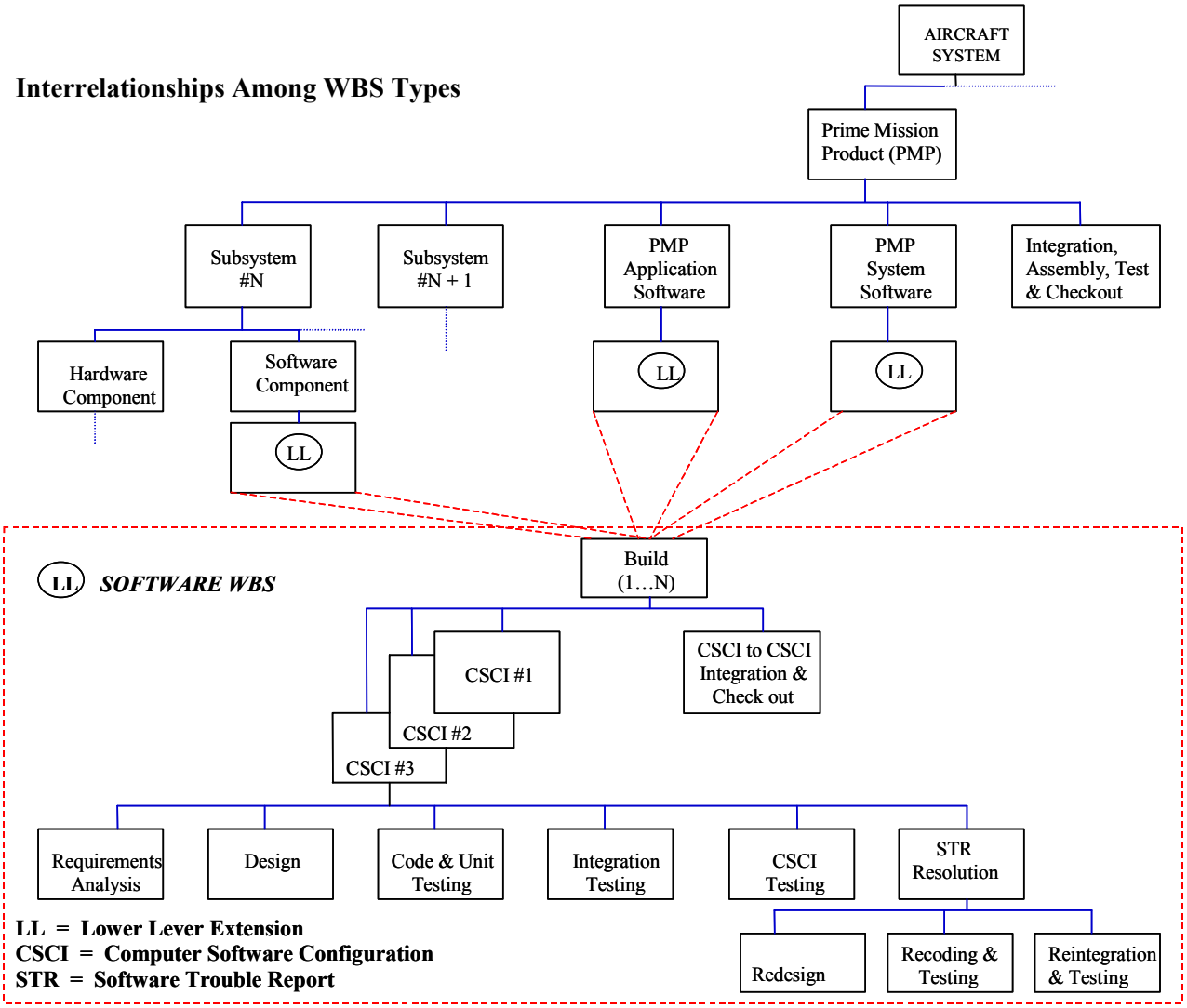


Figure 2-6: Project Summary WBS Example

Appendices E and F contain additional information on the software WBS.

For more and more DOD systems, software development will consume the majority of resources, schedule and cost while generating the bulk of program risk. Obscuring the software development effort by burying it deeply within the system WBS often as a sub component of much cheaper, lower risk hardware efforts can only aggravate these problems. It is important to keep these issues in mind when developing the Program and Contract WBS. The deeper the software is buried in the effort, the deeper the contractor must report and the greater the burden placed on the contractor's financial tracking and reporting system. Appendices E and F contain additional guidance on developing a software WBS.

2.7 Software Code Issues

How the code was produced also has a significant impact on cost and schedule. Several different types of software code development (new, reuse, modified, deleted, automatically generated, ported, and COTS) are discussed in the following sections. The different levels of effort required to implement and integrate each type of software means that each different type should be tracked separately. Failure to do so will make it very difficult to estimate the project and to update the estimate based upon actuals. Different types of software take varying amounts of effort, time and earned value to develop or modify for inclusion in a software development.

2.7.1 New Code

Code developed from scratch specifically for a project. New code is usually the most expensive to develop and most often underestimated in size. More time and effort, translated into more earned value, will be associated with each line of new code.

EVM Issues

1. Size is often underestimated. This results in overoptimistic costs and schedules with resulting low Cost Performance Index (CPI) and Schedule Performance Index (SPI).

2. Generally the most expensive type of code to produce. Every phase of software development must be implemented.

2.7.2 Reuse Code

Reuse code is previously developed (existing) code that will be integrated as-is into the system. The amount of functionality to be gained by reusing code is often overestimated. This results in higher cost and longer schedules when new code is used to satisfy requirements rather than reused code. Most of the effort with reused code involves integration into the system; therefore the costs, schedule and effort are much less than new code.

EVM Issues

1. The amount of functionality that can be gained through software reuse is often overestimated. This results in a reduction in the amount of reuse and an increase in new and/or modified code resulting in higher cost and longer schedules. CPI and SPI degrade.

2. Often cost and schedule overruns are experienced in development efforts based on software

reuse integration. These overruns can often be attributed to developer unfamiliarity with the code, poor documentation, and low quality of code.

2.7.3 Modified Code

Modified code was previously developed for a different system and is now being modified to meet the new systems requirements. Modified code is "usually" less expensive than new code, however this is not always the case. If the code is of low quality, poorly documented, or the developer is unfamiliar with the code, costs can rise steeply and may exceed the cost of developing new code. No matter how good the code being modified is, if the percentage of modification becomes large enough it becomes easier to develop new code than modify it.

EVM Issues

The amount of modified code that can be used in a system is often overestimated and/or the amount of modification the code will require is underestimated. This results in increasing cost and schedule for implementing and integrating the modified code or additional new code and subsequent degradation in CPI and SPI.

2.7.4 Deleted Code

Deleted code is software previously developed for another system that is being removed since it provides functionality not required by the new system. Parts of the remaining code will require retest, redesign and recoding in order to ensure that the remaining functionality and interfaces were not impacted by the deletions.

Deleting code is not a "No Cost" effort. If the code is poorly designed and documented, deletion can become very expensive because of the time and effort associated with retest and redesign to verify the remaining functionality. Often deleted code is included with modified code and not tracked separately.

EVM Issues

Similar to the problems with modified code. Often the amount of code to be deleted and the amount of testing after the deletions is underestimated as part of the modification effort. This leads to higher cost and schedule and lower CPI and SPI.

2.7.5 Automatically Generated Code

Software tools that interpret the software design and generate the source code and executable code produce automatically generated code. The generator may produce high quality code if the design it is given is correct, but it will also automatically generate defects caused by faulty software requirements and design. Testing will still be required.

Using automatic code generators does not eliminate the need for requirements analysis, design and testing and may eliminate less than 20% of the development effort compared to manual code generation.

EVM Issues

Automatically generated code does not produce free code. Requirements analysis, design and testing are still necessary for automatically generated code.

2.7.6 Converted/Ported Code

Another type of automatic code generator converts from one language to another, from one operating system to another, or both. This may be referred to as converted, translated or ported code. Usually done for maintainability reasons due to the old language, operating system or computer hardware no longer being supported.

If it becomes necessary to manually correct the source code generated by any of these methods, determining the cause and how to correct it can be very difficult due to the lack of commenting of the source code.

EVM Issues

Automatically converted/translated/ported code may require extensive manual corrections to get it to run in the new environment. However, if it was automatically generated without comments it is likely to be much more difficult, costly and time consuming to understand and modify or correct.

2.7.7 Commercial Off the Shelf (COTS).

COTS are essentially commercially acquired reuse code. When using COTS code, maintainability and availability are bigger concerns. The developer may decide to stop supporting the version or product used in the system or they may go out of business. In which case the system must either upgrade or change to a different tool if it wants to avoid having unsupported code. The integration costs for such COTS changes can be significant. The commercial viability and upgrade policies of a COTS vendor are serious considerations when selecting products. An established vendor with a long history, good future prospects, a large customer base and a policy of making their software upgrades backward compatible, may be a better choice even if their product isn't technically the optimum choice. Where a new company with a small customer base and questionable future prospects may be the wrong choice even if their product is technically the best choice. No matter how carefully planned for, a company going out of business or dropping a product line can cause severe schedule impacts on the program. If the survival of a vendor or product is questionable, select another vendor and product with a better outlook.

EVM Issues

1. Selection of a COTS product must not only consider the technical merit of the product but the commercial viability of the vendor. If the vendor's long-term prospects are questionable, a mitigation plan for replacing the product must be developed and additional funding to cover the risk built into the project.

2. The project plan must include plans for upgrading any COTS tools. Failure to do so is likely to result in unplanned for costs associated with product upgrades, integration and testing.

[1] Boehm, Dr. Barry, Software Engineering Economics, Prentice-Hall, 1981
[2] Abts, Chris, Boehm, Barry, Brown, A. Winsor, Chulanti, Sunita, Clark, Bradford K., Horowitz, Ellis, Madachy, Ray, Reifer, Donald, Steece, Bert, Software Cost Estimation With COCOMO II, Prentice Hall PTR, 2000
[3] Jones, T. Capers, Estimating Software Costs, McGraw-Hill, 1998
[4] Nelson, Maj Mike Nelson, Clark, James, Spurlock, Martha A., "Curing the Software Requirements and Cost Estimating Blues, The Fix Is Easier Than You Might Think", Program Manager Magazine, Nov/Dec 1999
[5] Jones, T. Capers, "Project Management Tools and Software Failures and Successes", Crosstalk, July 1998
[6] Solomon, Paul, Northrop Grumman Corporation, "Practical Software Measurement, performance Based Earned Value", Software Technology Conference presentation, 2 May 2000
[7] Guideline's for the Successful Acquisition of Software Intensive Systems, Version 3, 2000, US Air Force Software Technology Center, http://www.stsc.hill.af.mil/resources/tech_docs/

3. Software Metrics & Measures

The more common software metrics & measures that can be used as a basis to allocate earned value during the software lifecycle are described in this section. The advantages and disadvantages of using each metric or measure and how the metric or measure is used to allocate EV is also presented. This is not an all-encompassing list. It is a sampling of the most commonly used measures that are the most suitable or most often used in an earned value system by NAVAIR and its industry partners.

Note – Many of the examples in the following sections have tasks running much longer then 1 month. While it doe not specifically say in the examples, these tasks are rollups of many small tasks of 1 month or less length. This is to allow demonstration of the significant points of the example as briefly as possible, which is not possible if every low level task is discussed.

3.1 Requirements

3.1.1 Recommendation

Requirements as an EVM Measure – Good Recommended strongly as a basis for allocating BCWP

Requirements are an excellent measure for use in determining earned value measures since they are directly related to evaluating progress in implementing the functionality required by the system. Other software measures, even though they provide other critical project information, are further removed from the implementation of the requirements and thus reduce earned value accuracy.

A COMPREHENSIVE SOFTWARE DEVELOPMENT EXAMPLE USING REQUIREMENTS AS THE BASIS FOR DETERMINING EARNED VALUE IS PROVIDED IN APPENDIX J.

3.1.2 Overview & Description

Requirements are the primary cost driver of software development efforts. Software requirements tend to increase by 1–5% per month between the end of requirements analysis and the start of systems and integration testing, with the national average being about 2%[1]. Sometimes changes in requirements continue after testing begins. According to Capers Jones, approximately 20% of all defects in software are caused by poorly defined and contradictory requirements.

The requirements of a system are defined at many different levels. Requirements begin with the Mission Need Statement (MNS), a high level statement of operational capability. The desired and minimum acceptable levels of requirements of the new or proposed system are documented in the Operational Requirements Document (ORD). Under the current DoD standard for the

software development lifecycle[2], the following are deliverable products throughout phases of the software lifecycle that further decompose requirements:

1. Software Requirements Specification (SRS), Requirements Analysis Phase
2. Software Design Description (SDD), Design Phase
3. System Architecture and Requirements Allocation Description (SARD), Design Phase
4. Software Requirements Description (SRD), Design Phase
5. Software Test Description (STD), Testing Phases
6. Software Test Report (STR), Testing Phases

Any changes to the ORD, MNS or SRS which impact system requirements allocated to software will have an impact on the software requirements. In this case there will be a contractual impact since the scope of the effort is changing. Changes to the software requirements may also occur due to an improved understanding on the part of the contractor of the systems requirements, or a revision in how the software is to be implemented and thus can occur without a corresponding change in system requirements allocated to software. Software requirements changes, not caused by systems requirements changes, should not have a contractual impact.

There is no hard and fast rule that changes in software requirements due to improved understanding of the system by the contractor will not have contractual impacts. The improved understanding is usually gained through the developer consulting with Government program office representatives and users. If the Government representatives insist on interpretations of requirements that call for the most elaborate and complex implementation, or interpretations that may not be clearly or logically drawn from the system requirements as written, costs and schedule for the effort may climb beyond what the contractor could have reasonably foreseen. This is often the case when the Government provides a poorly written and incomplete set of requirements to the developer. In such situations where the Government cannot adequately define its requirements, additional time and funding must be made available for prototyping and/or additional spirals and releases during the development process or the Concept and Technology Development phase prior to milestone B must be extended to allow the Government to determine its requirements.

On the other hand, it is unreasonable for a developer to assume that there will be no changes to software requirements following the software requirements analysis phase and not build at least some capacity to handle such changes into the project plan. The US average is for about 2% changes in software requirements per month from the completion of software requirements analysis until the start of integration testing[3]. A project plan, which necessitates cost and schedule slip whenever a software requirement changes, is unexecutable. A reasonable amount of change should be built into the project plan based on the developer's and acquisition organization's prior project history.

Since requirements are the ultimate driver in determining software cost and schedule, they are also an excellent choice for determining earned value[4]. Requirements are applicable to all phases of the system and software development, which further increases their utility as a means of determining earned value. Additionally, requirements are directly related to producing the functionality the Government wants in a new system. Other metrics/measures that are indirectly

related to implementing the desired functionality can inject errors into the earned value calculations.

EXAMPLE:

Requirement: Build 100 miles of highway in 10 months for $10M
Contractor Estimate: 10,000 loads of fill, concrete, and other material required to complete requirement
Status: After 5 months, 30 miles have been completed and 5000 truckloads have been used. How is the project doing?

If the measure for EV was defined as follows:
Government Measure: # of miles completed Contractor Measure: # of truckloads used

Earned value would be reported as:

Measure: # of miles completed	BCWP = $5M and 60% behind schedule
Measure: # of truckloads used	BCWP = $5M and project on schedule

Earned value allocation must be based on the complete implementation of a requirement(s) in a development phase. Allocating earned value before the associated task(s) have been completed virtually always causes errors in earned value measures. This is much more likely to occur when measures other than requirements are used as the basis for allocating earned value.

If something other than requirements are used as the basis for EVM during the software development, care must still be taken to ensure the requirements planned for analysis/design/coding/testing in the phase in question are actually implemented. It is possible that in a multi build/release development effort, the estimated amount of code could be implemented but for other than the planned for requirements. In some cases, this could result in acceptable earned value results, but with programmatic impacts which will not be apparent until much later, such as:

1. Certain functionality may have been selected for implementation in a certain release in order to deliver it to the user by a certain date. Changing which requirements are implemented would impact meeting the users needs.

2. Suppose some of the requirements to be implemented in a release were essential for allowing software from another developer, reused software or COTS software to be integrated with the release. If these integration requirements were replaced by other requirements, the EVM might still look good initially, but it might have severe impacts on the integration effort and cause significant programmatic delays later in the effort.

IF REQUIREMENTS ARE NOT CONSIDERED WHEN DETERMINING EARNED VALUE, THEN EARNED VALUE WILL NOT REFLECT ACTUAL PROGRESS IN MEETING THE REQUIREMENTS.

Obviously if requirements are used as a means of taking earned value, some means of allocating earned value prior to when a requirement is fully implemented and completely tested is necessary. Earned value should be allocated based on the completion of the appropriate tasking for a phase for each requirement.

3.1.3 Phases Using Requirements for EVM

System and software requirements can be used in all software development phases as the basis for allocating earned value. The developer will need a methodology for estimating the BCWS for each requirement or logical grouping of requirements in each phase. Many developers do not break down cost and schedule for individual requirements for each phase. Attempting to estimate individually for every requirement adds an additional tracking and estimation burden on the developer, which may not be justified by resulting increases in EVM accuracy. It is also often impractical or impossible to estimate for individual requirements in isolation. An accurate estimate may only be possible when the requirements are considered in logical groupings, modules, Computer Software Components (CSC) or Computer Software Configuration Items (CSCIs).

If the number of requirements is sufficiently large and sufficiently detailed, the developer may choose to assume that the amount of effort required for all requirements to be implemented is equal. This simplifies and reduces the effort required to determine BCWS for each software requirement. This assumption increases in validity, as the requirements are decomposed to the level of software requirements as documented in the SRD. These SRD level software requirements are roughly equivalent to testable requirements, which is one proposed method of determining software size[5]. Assuming an equal level of effort for system requirements is riskier due to the wider range of effort it can take to implement these higher-level requirements.

In order to utilize requirements as the basis for taking earned value, the developer must have a requirements traceability system able to track requirements from at least the level of the system requirements through software requirements, builds, CSCIs, design, code & unit test, and test procedures for all test phases. The ORD and MNS requirements should also be traced to the SRS systems requirements. Such traceability is essential to insuring that the Government's required functionality is fully implemented and tested in the system and should already be done by any mature developer. In order to reach SW-CMM® Level III[6] or CMMI® Level III[7] a developer must have such a requirements traceability program implemented. Since NAVAIR INST 5234.1[8] mandates that software developers for NAVAIR be a level III organization, such traceability will not be an additional burden on the developer in order to use requirements as the basis for earned value.

3.1.3.1 Software Requirements Analysis Phase

Software requirements analysis is the decomposition of systems requirements into more detailed software requirements. Software requirement must be both detailed enough so that the software design can be unambiguously generated from them and so that test procedures can be developed to verify them. During the Software Requirements Analysis Phase, earned value would be allocated based on how many of the systems requirements allocated to software (as identified in

the SRS and the Software Architecture and Requirements Allocation Description) for which the software requirements analysis had been completed.

If peer reviews of the requirements were performed, the successful completion of the requirements peer review ***ALONG WITH THE CORRECTION OF ANY IDENTIFIED PROBLEMS*** could be established as the point where all earned value (BCWP) for the requirement(s) would be allocated. Peer reviews of software requirements are called for in organizations achieving SW-CMM® Level III[6], or CMMI® Level III[7]. NAVAIR Instruction 5234.1[8] requires all NAVAIR ACAT I, IA, II, III and IV programs to be certified at SW-CMM® Level III or its equivalent. Using successful peer review completion as the milestone for allocating earned value during the software requirements analysis phase will not place an additional burden on the developer. Using something other than successful completion of peer review to determine when the requirements analysis is complete will increase subjectivity and reduce earned value accuracy.

EXAMPLE:

Project Requirement: 1000 system requirements in 10 months for $1M
Assumption: Each system requirement takes the same amount of effort to perform software requirements analysis.
Status: After 8 months, 900 system requirements have been completed with all noted defects from peer review corrected.

What is the earned value for this effort?

$$BCWP = (\$1M) \times 900 / 1000 = \$900K$$

CWP = $850K (Monthly expenditures have been higher than planned.)
BCWS = $800K ($100K per month for 8 months.)

$$CPI = BCWP / ACWP = \$900K / \$850K = 1.056$$

$$SPI = BCWP / BCWS = \$900K / \$800K = 1.125$$

As stated previously, it is risky to assume that software requirements analysis for all systems requirements will take the same amount of effort. The developer may choose to assign different BCWS amounts to different systems requirements based on varying amounts of effort required to perform software requirements analysis, this increases the overhead required to implement the EVM system due to additional tracking effort.

Refer to the following for additional information:
1. Executive Summary
3.2.3.3.1 Software Requirements Analysis Phase (EVM & Function Points Metric)
3.1.4 Deferred Functionality or Requirements
3.1.5 Capacity & Performance Requirements Issues
3.1.6 General Requirements Issues

3.1.3.2 Software Design

In software design the software requirements are decomposed into a detailed architecture and design from which the code can be directly, and unambiguously produced. During the Software Design Phase, both high level and detailed, earned value would be allocated based on how many of the software requirements (as identified in the SRD(s)) the design has been completed for. If design peer reviews are utilized, completion of the design and allocation of the BCWP for that requirement could occur when the peer review and ***ANY NOTED DEFECTS HAD BEEN CORRECTED***. Allocation of BCWP may or may not be the same for all software requirements depending on the developer's evaluation of the amount of effort required to design different software requirements. Peer reviews of software design are called for in organizations achieving SW-CMM® or CMMI® Level III. Using something other than successful completion of the peer review to determine when the software design is complete will increase subjectivity and reduce accuracy of the earned value.

EXAMPLE:

Project Requirement: 400 system requirements in 8 months for $1M

Assumption: Each software requirement takes the same amount of effort to develop a design.

Status: After 4 months, design for 175 requirements has been completed with all noted defects from peer review corrected. What is the earned value for this effort?

BCWS = $500K

$$BCWP = (\$1M) \times 175 / 400 = \$437.5K$$

ACWP = $450K (Actual expenditures per month are less than planned)

$$CPI = BCWP / ACWP = \$437.5K / \$450K = .972$$

$$SPI = BCWP / BCWS = \$437.5K / \$500K = .875$$

Status: After 8 months there is improved understanding of the system requirements which results in an increase of total software requirements to 440. No contract modification is required to account for the increase. Design for 390 requirements has been completed with all noted defects from peer review corrected. What is the earned value for this effort at this point?

BCWS = $1M

$$BCWP = (\$1M) \times 390 / 440 = \$886.4K$$

ACWP = $1,003K

$$CPI = BCWP / ACWP = \$886.4K / \$1{,}003K = .884$$

$$SPI = BCWP / BCWS = \$886.4K / \$1M = .886$$

Status: At the end of 9 months, design for all 440 requirements has been completed.

What is the total earned value for this effort?
BCWS = $1M
BCWP = $1M
ACWP = $1,101

$$CPI = BCWP / ACWP = \$1M / \$1{,}101K = .908$$

SPI = 1.

The developer's plan did not take into consideration the possibility of an increase in software requirements. This unexpected growth may account for part of the cost and schedule overrun. It also appears that the planned productivity was higher than was actually achieved.

Refer to the following for additional information:
1. Executive Summary
3.2.3.3.2 Software Design Phase (Function Points Metric)
3.3 Modules (EVM & Modules Metric)

3.1.3.3 Code & Unit Test (C&UT) Phase

During C&UT source code is generated from the software design. Developers then conduct low-level unit testing to verify that the design has been correctly implemented. During the C&UT Phase, earned value would be allocated based on how many of the software requirements (as identified in the SRD(s)) for which the C&UT had been completed. If peer reviews are utilized in this phase, completion of a peer review and allocation of the BCWP for that requirement could occur when the peer review and ***ANY NOTED DEFECTS HAD BEEN CORRECTED***. Peer reviews of software requirements are called for in organizations achieving SW-CMM® or CMMI® Level III. Using something other than successful completion of the peer review to determine when the C&UT is complete will increase subjectivity and reduce accuracy of the earned value.

EXAMPLE:

Test Plan Requirement: 200 software requirements in 5 months, BCWP = $550K.
Assumptions: 1) Each software requirement takes the same amount of effort to perform C&UT. 2) Task will be evenly spread over the 5-month; BCWS = $110K per month.
Status: At the end of 2 months, 85 defects in the software requirements discovered through C&UT and peer reviews have been corrected. Due to improved understanding of system requirements, total software requirements increased to 206 requirements. No contract mods are required. What is the earned value for this effort?
BCWS = $220K

$$BCWP = (\$550K) \times 85 / 206 = \$226.9K$$

ACWP = $230K

$$CPI = BCWP / ACWP = \$226.9K / \$230K = .987$$

$$SPI = BCWP / BCWS = \$226.9K / \$220K = 1.031$$

Status: At the end of 5 months, 210 defects in the software requirements discovered through C&UT and peer reviews have been corrected. Total software requirements increased to 210 due to an improved understanding of system requirements. No contract mods are required. What is the total earned value for this effort?

BCWS = $550K
BCWP = $550K
ACWP = $545K

$$CPI = BCWP/ACWP = \$550K/\$545K = 1.009$$

SPI = 1.0

Refer to the following for additional information:
1. Executive Summary
2.7 Software Code Issues
3.2.1 Code & Unit Test Phase (SLOC Metric)
3.2.3.3.3 Code & Unit Test Phase (Function Points Metrics)

3.1.3.4 Test Phases

There are a variety of test phases following C&UT which seek to verify that the system and software requirements have been correctly implemented via the execution of formal test procedures. IEEE/EIA 12207 identifies the following test phases: software integration testing, software qualification testing (often referred to as Formal Qualification Testing (FQT) in DoD), Systems Integration Testing and System Qualification Testing. Avionics systems being delivered to the fleet will also go through a variety of flight testing, Developmental Testing (DT) and Operational Testing (OT). During the Software and Systems Test Phases, earned value would be allocated based on how many of the systems or software requirements (as identified in the SRD(s)) had been ***SUCCESSFULLY*** tested. ***SUCCESSFUL TESTING FOR A REQUIREMENT MEANS THAT ALL ASSOCIATED TEST PROCEDURES HAVE EXECUTED TO COMPLETION AND NO DEFECTS PREVENTING THE EXECUTION OF THE REQUIREMENT HAVE BEEN GENERATED***. For software integration testing, software requirements will be most appropriate for taking earned value. For systems testing, DT and OT the systems requirements may be more appropriate. The appropriate type of requirement for taking earned value is dependent upon the type of requirements used as the basis for developing test procedures for the test phase in question.

As part of the preparation for a test phase, a test plan and test procedures must have been developed and peer reviewed. ***AS PART OF THE PEER REVIEW, THE TEST PROCEDURES MUST BE CHECKED TO ENSURE THAT THE PROCEDURES TEST ALL REQUIREMENTS AND THAT EACH TEST PROCEDURE IDENTIFIES THE REQUIREMENTS IT TESTS.*** This information is essential to determining which requirements have been successfully tested and for which earned value can be allocated.

Successful completion of a test procedure is not necessarily the same as no defects occurring during the test. See section 3.5 for a discussion of the different defect priorities. All priority 1 and 2 defects must be corrected prior to Operational Testing and the impact of all priority 3

defects documented[9]. **ALSO, A PROGRAM SHOULD CONTRACTUALLY IDENTIFY SPECIFIC SOFTWARE QUALITY TARGETS.** Thus priority 4, 5 and to a lesser extent priority 3 defects can occur during testing without considering the test procedure to have failed.

Refer to the following for additional information:
1. Executive Summary
3.1.3.5 Software Rework (EVM & Requirements Metric)
3.2.3.4 Test Phases (EVM & Function Points)
3.4 Test Procedures/Cases (EVM & Test procedures/Cases Metric)

3.1.3.5 Software Rework

Software rework is the correction of defects. These defects may be in the requirements, design and other documents, or in the code itself. A defective requirement will cause defective design and defective code; a defective design will cause defective code. Obviously the sooner a defect is detected and corrected, the less the cost since it will not snowball into later development phases. Cutting corners on quality processes in early development phases results in more defects which are not detected until much later in the development with resulting significant increases in development costs.

REWORK MUST BE INCLUDED IN THE SCHEDULE FOR ANY SOFTWARE DEVELOPMENT PROJECT. It is unreasonable to assume that there will be no defects detected in any of the requirements, design or code. Additionally, if such rework phases are not planned for, it can cause severe problems to the earned value system when it is attempted to determine how to implement it at the spur of a moment. Programmatically, any project plan that does not include time for rework is unexecutable and questions the maturity of the developing organization. The developer must take into consideration that some percentage of the requirements will not pass testing. The rework must not only include time to correct the flaw in requirements, design and/or code that caused the problem, but also to retest the corrected software. In a multi release/build development, this may mean that some or all of the failed requirements will be rolled into the next build/release. All of this must be taken into account in the project plan.

Rework should be planned and tracked in separate work packages from the initial development of requirements, design and code. In planning incremental builds, all builds must include budget and schedule for rework of requirements, design and code to correct defects that were found in the current and previous builds. To ensure adequate budget and period of performance, the planning assumptions for rework should include the planned rate or number of defects expected and the budgeted resources to fix the defects. Failure to establish a baseline plan for rework and to objectively measure rework progress has caused many projects to get out of control[4].

EXAMPLE:

Project Requirement: 1000 software requirements in Build A.
Assumption: None of the software requirements are on the critical path**.**
Program Plan: Based on the developer's experience, 10% of these requirements will fail testing, 5% will be rework, 5% will be deferred to Build B. The program would thus

plan the rework phase for build A to include resources and schedule to allow the rework of defects impacting 50 requirements. At the end of the rework phase in Build A, 95% or 950 of the total of 1000 software requirements would be correctly implemented and the 50 incorrectly implemented requirements are deferred to build B.

Scenario 1: More than 100 requirements failed testing. If the predicted performance in correcting each defect is achieved, then one of the following will occur:

1. More time and resources will be required in the rework phase in build A, which will drive ACWP up, CPI and SPI down.
2. Less than 950 requirements will be correctly implemented in build A and more than 50 requirements will be deferred to build B. This means the BCWP will be less than BCWS at the end of Build A with the difference deferred to build B which will also reduce CPI and SPI. ***DO NOT REPLAN WHEN MORE THAN THE PLANNED AMOUNT OF FUNCTIONALITY IS DIFFERED TO A LATER PHASE.*** Such a replan will hide the schedule and cost slips caused by excessive deferral of functionality.

Scenario 2: Less than 100 requirements failed testing. If the predicted performance in correcting these defects is achieved, then one of the following will occur:

1. Less time and resources are needed than was planned to correct all the defects except for the 50 planned for deferral to build B; rework will be finished sooner than planned. This will reduce ACWP; increase BCWP, which will drive CPI and SPI up.
2. The developer will use the planned time and resources for rework in build A, resulting in more than 950 requirements being implemented in build A and less than 50 being deferred to build B. This will result in BCWP being greater than BCWS at the end of Build A which will also drive up CPI and SPI. This will require some replanning to adjust the BCWS since BCWP cannot exceed BCWS. In any case the project will be below cost and ahead of schedule in the rework phase.

Scenario 3: The developer plans to implement 950 new software requirements in Build B, along with the 50 requirements that failed testing in Build A and were deferred to Build B.

Again, they may plan on only successfully implementing 95%, or 950 software requirements in order to take 100% of the planned earned value at the end of build B.

Obviously at some point all the defects must be corrected, or at least most of them. All software contains some defects when released. Additional time may be included in the final release to clean up defects or one or more releases may be planned at the end of the development for defect correction.

Usually the amount of time required to correct defects is based on historical data from previous projects. There is a very wide variance in the amount of time required to fix individual defects,

however, for large systems with resulting relatively large numbers of defects averages tend to work out.

Defects are also defined by priority; see section 3.5 for additional discussion of defect priorities. While it would be nice if software could be released defect free, this is unfortunately impossible. A disciplined mature development process can significantly reduce the number of defects but not eliminate them. Priority 1 & 2 defects must be corrected prior to commencing Operational Evaluation (OPEVAL) since they prevent the execution of essential requirements. Priority 3 defects have workarounds but their impacts on the system must be documented prior to entering OPEVAL[9]. If there are too many priority 3 defects, this can have a significant impact on the operator's ability to perform the mission and thus make it unlikely the system will pass OPEVAL. Priority 4 & 5 defects are not required to be corrected prior to OPEVAL[9]. ***A SOFTWARE DEVELOPMENT PROGRAM SHOULD CONTRACTUALLY ESTABLISH QUALITY CRITERIA DEFINING THE MAXIMUM NUMBER OF DEFECTS OF DIFFERENT PRIORITY FOR THE EFFORT***. From the perspective of using requirements for the allocation of earned value, priority of the defects determines which defect must be corrected in order for the software requirements to have been considered successfully implemented.

In the previous example, priority 4 and 5 defects are likely to be completely ignored as far as EVM is concerned since they have minimal or insignificant impact on requirements implementation. Determination of which and how many priority 3 defects must be corrected before the BCWP for a requirement can be earned is at least partially determined by the contractual quality requirements.

EXAMPLE:

Assumptions: Following testing in Build A there are:

1. 10 priority 1 and 2 defects affecting 5 requirements,
2. 100 priority 3 defects affecting 50 requirements, and
3. 100 priority 4 and 5 defects identified.

In this situation all of the priority 1 and 2 defects must be corrected. If they are not, the BCWP associated with those requirements cannot be earned and the software cannot go to OPEVAL and will fail if it does.

Assumption: The system's quality requirements are such that only 50 of the 100 priority 3 defects need to be corrected. In this case if all of the priority 1 and 2 defects are corrected, and 50 of the priority 3 defects are corrected, all of the BCWP for build A would be earned. The remaining priority 3, 4 and 5 defects might only be corrected if it was convenient and easy to do so as a result of other work. It is unlikely there would be any planned effort to correct priority 4 and 5 defects unless the contractual quality requirements made this necessary.

NOTE – IF QUALITY REQUIREMENTS IDENTIFYING THE MAXIMUM NUMBER OF DIFFERENT PRIORITY DEFECTS ARE NOT CONTRACTUALLY SPECIFIED, EARNED VALUE ACCURACY WILL BE REDUCED, MAKING IT POSSIBLE TO EARN BCWP WITH LARGE NUMBERS OF DEFECTS IN THE SOFTWARE THAT WILL MAKE SUCCESSFULLY PASSING OPEVAL UNLIKELY. THIS IS TRUE NO MATTER WHAT MEASURE IS USED TO ALLOCATE BCWP.

From an earned value point of view, the most important aspect of rework is the correct implementation of software requirements; however an estimate of the time and resources required to perform rework will probably be based on estimates of the number of defects from historical data on previous projects or actual data from earlier rework phases in the current development. While defect estimates may be used for estimation purposes, they are not effective for use in determining BCWP since they are not always directly related to requirements. In some cases several defects may need to be corrected in order to correct the implementation of a single software requirement, in other cases correcting a single defect will correct the implementation of multiple software requirements.

In order to use requirements as the basis for determining BCWP during rework, an effective requirements tracking system must be in place, which traces individual defects to the requirements they affect and identifies which defects should be corrected based on priority and contractual quality requirements. This should not place a significant additional burden on the developer since such a requirements tracking system should already be in place for any SW-CMM® or CMMI® Level III organization.

Refer to the following for additional information:
1. Executive Summary
3.2.3.3.5 Rework (EVM & Function Points Metric)
3.5 Software Defects (EVM & Software Defects Metric)

3.1.4 Deferred Functionality or Requirements

When functionality is deferred, requirements intended for implementation as part of a specific build are delayed until a later build. If systems or software requirements intended to be implemented in build A, do not have their design completed during the design phase, they cannot be coded and subsequently tested in Build A (if they are there will be severe quality problems), they must be deferred for completion to a later build. Requirements which don't have their code completed in Build A cannot be tested in Build A. They must be deferred for completion to a later build. Deferring functional requirements has the following major impacts:

1. If all the requirements planned for a phase are not completed, then the earned value for these deferred requirements cannot be earned as part of the build.

2. The phase and/or build the requirement is deferred to will require additional time and resources to complete its planned requirements and the deferred requirements. ***THE EARNED VALUE ASSOCIATED WITH THESE DEFERRED REQUIREMENTS, WHICH WAS NOT EARNED IN THE PHASE OR BUILD IT WAS DEFERRED FROM, WILL INSTEAD BE EARNED IN THE PHASE AND/OR BUILD IT WAS DEFERRED TO.*** Unless of course it's deferred again.

3. ***IF THE DEFERRED REQUIREMENT IS ON A CRITICAL PATH FOR THE PROJECT, IT CAN HAVE A MUCH GREATER IMPACT ON THE PROJECT THAN IS REFLECTED BY THE AMOUNT OF ADDITIONAL TIME AND EFFORT REQUIRED TO COMPLETE IT***. The implementation of a software requirement will be on the critical path if its delay will cause:
   a. A delay in implementation of other functionality.
   b. A delay in development on another project.
   c. A milestone for product delivery to the operational forces to be missed.

4. Delay in implementing critical path software requirements can rapidly snowball into much larger cost and schedule impacts due to the resulting delay they cause on the implementation of other functionality and projects. Thus for critical path requirements deferring them may be an unacceptable option due to these critical program schedule disruptions.

5. Although requirements may be deferred to a subsequent build, the earned value must continue to show a behind schedule condition. ***THE DEFERRED EFFORT SHOULD NOT BE REPLANNED*** beyond the current month[4].

Defects are one of the most common causes of requirements deferrals, but others exist such as:

1. Underestimation of technical risk, or it's a lot harder than originally thought.
2. Overestimated productivity, or it's going to take more people than originally planned.
3. Staffing shortfall, or all the people planned for aren't available.
4. A resource isn't available. GFE or COTS isn't available or doesn't show up on time, thus causing a delay in the development.
5. Funding levels or profile changes.
6. Changes to systems requirements.

A cursory look at the issues that can cause requirements deferrals indicates that they are all program risks. A mature software development organization can build into its project plan time and resources for the various software development tasks, including rework based on historical data from previous projects. Basing the project plan on historical performance data is a method of mitigating the risk of an overly optimistic budget and schedule by basing it on what has been achieved previously. Unfortunately risks often identify potential problems on which either no data is available or the data is very scant, thus making it very difficult to predict what will happen or if the risk will occur. In such cases actual data from the project may be the only source of information on the likelihood and impact on the program of these risks. This is why it is so essential that an effective measurement program, including earned value, must provide visibility into significant program risks.

If an effective risk management, measurement and earned value program is combined, it can serve to identify the occurrence of the risks at an earlier date when it is more likely that effective corrective action which minimizes perturbations to the program plan can be made. ***REQUIREMENTS DEFERRAL IS ALWAYS THE RESULT OF A RISK OCCURRING. THE MORE EFFECTIVE A PROGRAM IS AT MANAGING AND TRACKING ITS RISKS, THE LESS FUNCTIONALITY OR REQUIREMENTS WILL BE DEFERRED.***

An effective earned value system must account for deferred functionality if it is to accurately reflect program status and progress. In order to do this, the system or software requirements planned to be implemented in each software development artifact or phase must be considered in determining earned value. ***NO MATTER WHAT SOFTWARE MEASURES ARE USED TO DRIVE EARNED VALUE, REQUIREMENTS MUST ALSO BE USED IF ACTUAL PROGRAM STATUS IS TO BE DETERMINED.***

3.1.5 Capacity & Performance Requirements Issues

Capacity and performance requirements define functionality that impacts how a large percentage of other software requirements if not all of them are implemented. Failure to meet these requirements can have a significant negative impact on meeting the systems cost and schedule objectives and can require significant redesign of the system hardware and/or software. In order for EVM to be effective, it must be able to reflect the negative cost and schedule impact on the program if such problems arise.

3.1.5.1 Description

Capacity requirements specify the maximum amount of available processing resources that can be used by the software application being developed. These processing resources include computer processing unit (CPU) capacity, random access memory (RAM), both dynamic and static memory, hard drives and other non RAM static memory, interface or bus throughput and other computer resources. Usually for all of these resources a maximum percentage of the total capacity of the resource is identified which the software can utilize, usually 50% for new developments or major upgrades. For example:

1. No more than 50% of the total CPU capacity will be utilized by the system's software.
2. No more than 50% of the RAM is utilized by the system's software.
3. No more than 50% of the available interface/bus throughput is used by the system's software.

Many systems include multiple CPUs, blocks or types of memory, interfaces or buses, hard drives and other computer resources both in a specific computer and also as part of a distributed network in the system. Capacity requirements apply individually to each of these resources in the system. If there are multiple CPUs in the system and there is a requirement that no more than 50% of the processing capacity of the CPUs be used, this applies to each individual CPU and not the average processing capacity of all CPUs. If a CPU exceeds this requirement, it will complicate future maintenance and upgrade of the system due to a lack of processing capacity in this component. In a worse case scenario, it will act as a bottleneck on the entire system due to lack of processing capacity in the CPU. The same applies to RAM, interfaces/buses, and other computer resources.

Performance requirements, or real time requirements, mandate some type of response requirement on the system, usually related to reacting within a specified period of time to some input or event. For example:

1. Within .25 seconds of receiving operator input the system will provide operator feedback that the input has been received.
2. The system will be able to process 8 HZ navigation data with no loss of data.

3. The system will be able to receive and process up to 1000 radar contacts per second and process with no loss of data.

Functional requirements are requirements that specifically identify a capability that must be implemented in the software, such as:

1. Display on the tactical plot operator specified latitudes and longitudes.
2. Color all hostile targets on the tactical plot red.
3. Store the location, speed, altitude and course of all tracks and targets with an accuracy of 1 meter.

Failure of the system to meet a capacity or performance requirement will also affect the implementation of functional requirements.

EXAMPLES:

Example 1 - Assumptions:

1. There are 100 functional requirements that are either completely or partially implemented in software executing on CPU "A".
2. There is a performance requirement that no more than 50% of the processing capacity of CPU "A" can be utilized by the systems software.
3. The software running on CPU "A" is using 80% of the processing capacity

This indicates that all 100 functional requirements implemented on the CPU are at least partially incorrect since their implementation has resulted in the CPU utilization requirement for CPU A being exceeded by 30%.

Example 2 - Assumptions:

1. There is a performance requirement that the system respond within .25 seconds to operator inputs.
2. There are 100 functional requirements related to operator inputs that are not meeting this performance requirement.

This indicates that these operator input related functional requirements are at least partially incorrect since there implementation does not meet the performance requirement for a response to operator input within .25 seconds.

There are several different ways that such capacity and performance problems could potentially be corrected, such as:

1. **REDESIGN THE SOFTWARE TO MAKE IT MORE EFFICIENT**. This option can be extremely expensive, since it could require an extensive rework of the software design and also possibly the hardware design along with recoding and testing.

2. **RELAX THE CAPACITY OR PERFORMANCE REQUIREMENT.** This will likely increase the cost of future upgrades or maintenance to the system due to a shortage of the resource for which the requirement was relaxed.

3. **INCREASE THE CAPACITY OF THE RESOURCE THAT IS NOT MEETING THE REQUIREMENT.** In a COTS based system this *"may"* be an attractive option. It *"may"* be much cheaper to replace a CPU with a more powerful model, add more RAM, use a wider bandwidth bus, etc., than performing an extensive software redesign, coding and testing effort. Hardware upgrade costs could include rugadizing the new hardware components to insure it meets operational requirements. It will also require that the software be tested on the new hardware to insure that it is functionally equivalent. Software modifications may be necessary to get the software to operate on the upgraded hardware. Thus in some cases hardware upgrades may actually be more costly and time-consuming than software mods.

4. **SOME COMBINATION OF THESE OPTIONS.**

Each of these options will have different impacts on cost and schedule for the system and they may all not be practical for every situation.

3.1.5.2 Technical Performance Measurements (TPM)

TPMs are used to measure progress in achieving the technical objectives of the system. TPMs are phased over time in order to judge the progress in meeting Key Performance Parameters (KPP), which are usually associated with measurable performance or capacity requirements, such as discussed in section 3.1.5. Appendix H contains additional information on TPM as applied to software.

3.1.6 General Requirements Issues

Like capacity and performance requirements, general requirements can affect the implementation of large numbers of other functional software requirements. In the cases of the above three requirements, they will impact how all requirements implementing the operator interface are implemented. Unlike capacity and performance requirements, there is no single measurement that can be made to determine if general requirements are being implemented. Therefore, unlike capacity and performance requirements, it is difficult or impossible to specifically allocate earned value based on the implementation of such general requirements. Instead, implementation of the functional requirements should not be considered complete for any of the development phases unless their implementation also meets the appropriate general requirements. Thus if the code which implements a specific functional requirement for the user interface does not implement a general requirement for error checking of the operator inputs, the code for the specific functional requirement would not be considered complete and the earned value for code and unit test of the specific functional requirement would not be allocated until the error checking general requirement had been implemented. A general requirement could be partially implemented if it is included in some of the functional requirements implementation to which it applies and not to others.

[1] Jones, T. Caper, Estimating Software Costs, McGraw-Hill, 1998
[2] IEEE/EIA 12207, Software Life Cycle Process
[3] Estimating Software Costs, T. Capers Jones, McGraw Hill, 1998, pp 25, 40, 148, 193
[4] "Practical Software Measurement, Performance-Based Earned Value" http://www.testablerequirements.com/Articles/solomon.htm, CrossTalk, September 2001, Paul Solomon, Northrop Grumman Corporation , http://www.stsc.hill.af.mil/crosstalk/2001/09/index.html
[5] Sizing Software Using Testable Requirements, http://www.testablerequirements.com/
[6] Key Practices for the Capability Maturity Model Version 1.1, CMU/SEI-93-TR-025, http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.025.html
[7] CMMI Models, http://www.sei.cmu.edu/cmmi/models/
[8] NAVAIR INSTRUCTION 5234.5 NAVAL AIR SYSTEMS COMMAND METRICS FOR SOFTWARE INTENSIVE PROGRAMS, 30 September 02, https://directives.navair.navy.mil/index.cfm
[9] SECNAVINST 5000.2B, Part 3 Program Structure Paragraph 3.4.3.1 Navy Criteria for Certification, subparagraph 17, and 3.4.3.2 Marine Corp Criteria for Certification, subparagraph 17. http://neds.nebt.daps.mil/5000.htm

3.2 Size

Software size is the primary driver of cost and schedule. Initial size estimates, especially in the early development phases before the software requirements are fully defined, are often incorrect. Unfortunately, in the vast majority of cases, these size estimates are low compared to actual implementation size. The tendency of many software developers toward optimistic size estimates results in the actual software size often being much larger than original estimates, even when the actual implemented requirements for the system are much less than originally planned. ***Estimated size is based on planned requirements while the actual size is dependent on what requirements are implemented. They are not the same thing!***

Software size is usually given in Source Lines of Code (SLOC) or Function Points (FPs). SLOC, FPs, and other software size measures serve a purpose similar to weight and dimensions for an aircraft and are generated from the software requirements just as aircraft weight and dimension estimates are a result of its requirements. Size estimates are often based on the requirements for a module, Computer Software Configuration Item (CSCI) or build and then used as the basis for the estimates for the software development phases.

3.2.1 Source Lines Of Code (SLOC)

3.2.1.1 Recommendation

SLOC as an EVM Measure – Poor Not recommended as a basis for allocating BCWP

Because of the inaccuracy of SLOC estimates, SLOC is a poor measure to use in determining BCWP in earned value. The inaccuracy of SLOC estimates further degrades the correlation between the amount of SLOC implemented and the software requirements implemented. If SLOC are used as the basis for determining BCWP, they are only appropriate for use in the code & unit test phase and every effort must be made to continually update the SLOC estimate based on current data to make sure it is as accurate as possible. Additionally, requirements must also be monitored in order to verify that the software requirements planned for implementation are actually completed. If SLOC is used as the basis for allocating EVM, the SLOC count should be updated continuously as new data becomes available. ***SLOC IS ONLY USEFUL DURING THE CODE AND UNIT TEST PHASE. EVEN IN THE CODE & UNIT TEST PHASE, IF USED FOR DETERMINING BCWP IT IS LIKELY TO YIELD VERY INACCURATE RESULTS.*** It may be of some utility in code rework in order to correct defects, but applying it to take earned value for such rework is an even more difficult proposition than its use during code and unit test.

3.2.1.2 Overview & Description

The two main methods of counting SLOC are: (1) physical lines of code, counting each individual line, and (2) logical lines of code, counting only executable lines and declarations. There are a wide variety of different procedures and tools available for implementing these

SLOC counting methods which result in wide variances in SLOC totals. It is essential in any software development that all parties involved understand how SLOC is being counted on the project. It is also essential when comparing software developments from different organizations that the SLOC counting methodology is understood for all of the organizations. The same SLOC counts generated with different SLOC counting rules are not the same size nor will they require the same cost and effort to implement.

The primary problem with using a SLOC based EVM approach is the increase in initial size estimates, often low, over the development cycle. These low estimates are often the result of a poorly defined and undisciplined SLOC estimation method. It can also be the result of an inadequate requirements definition and poor requirements control during the life of the program. The best SLOC estimates will occur when performed by an experienced team of software engineers with access to:

1. SLOC sizing data from other software efforts of similar functionality,
2. A well-defined set of software requirements and
3. A disciplined methodology for developing the SLOC estimates from the requirements.

The accuracy of the SLOC estimate will degrade if all of these features are not available. Unfortunately, during the early phases of the system before the software requirements are fully defined, the accuracy of any SLOC estimate will suffer. This is true of any size estimation methodology.

Software developers often overestimate their productivity and underestimate the amount of code required to implement planned functionality. Basing earned value on an estimated SLOC count that is low results in a high CPI and SPI.

EXAMPLE:

Estimated Effort: 10K SLOC in 3 months to implement a set of requirements
Status: After 3 months, 10K SLOC has been completed, the software has *fully implemented* all planned functionality. How will earned value be credited for the effort?

EV = 100%, CPI = 1.0, and SPI = 1.0

Now, assume that SLOC was originally underestimated.
Status: After 3 months, 10K SLOC has been completed, the software has *implemented only half* of the planned functionality. How will earned value be credited for the effort?

EV = 100%, CPI = 1.0, and SPI = 1.0

Obviously this does not reflect the actual status of the development effort. Earned value should be reported as:

EV = 50%, CPI = .5, and SPI = .5

Scenario 1: Assume that the developer continues to work on the coding effort until all planned functional requirements are implemented. In this case, since earned value is based on the amount of SLOC completed, EVM indicators would show that the

project is on schedule and within cost until the planned amount of SLOC had been completed. At that time, BCWS = BCWP = ACWP, 100% EVM allocation. However, the developer will continue to expend hours to complete the work package. Thus EVM will have given no warning of a problem until the project reaches and exceeds the estimated SLOC.

Scenario 2: Assume that the developer stopped development once earned value equaled 100%. In this case there would be no indication of a problem since basing earned value on SLOC gives no indication of how much functionality has been implemented. The problem would not become obvious until testing when large numbers of defects would be discovered due to the incomplete functionality of the code. However this would have significantly delayed the discovery of the problem.

Use of SLOC as the basis of earned value should also be avoided since it is difficult to account for revised SLOC estimates without resorting to rebaselining. Instead, earned value should be based on the percentage of estimated SLOC currently completed rather than total SLOC estimated. Even using percentage of completed SLOC is far from perfect. It can result in the BCWP being reduced from one reporting period to the next.

EXAMPLE:

Estimated Effort: 10 month linear schedule, BAC = $1M, BCWS = $500K, ACWP = $500K

Status: In Jan 01, month 5, the estimated SLOC to complete a task is 10K. Amount of completed SLOC is 5K. Productivity (SLOC/hr) achieved has been the same as the prediction used to develop the Performance Measurement Baseline (PMB), and staffing levels have matched the plan. What is the earned value at this point?

BCWP = $500K, CPI = 1.0, AND SPI = 1.0.

Status: In Feb 01, the current SLOC estimate to complete the task increased to 15K. Amount of completed SLOC is 6K.

NOW, 40% OF THE CURRENT SLOC ESTIMATE IS COMPLETE.

Finally, assume that ACWP and BCWS have both increased to $600K. What does this Indicate for Feb 01?

THE BCWP HAS DECREASED TO $400K, CPI AND SPI HAVE DECREASED TO .66.

Status: In Aug 01, 15K SLOC have been completed. What is the earned value to date?

ACWP = $1.5M, BCWP = $1M, BCWS = $1M, CPI = .66

We assume that all the originally planned functionality has been implemented, however, this cannot be verified from the SLOC alone. Other requirements measures must also

be evaluated to determine if this is the case. It is possible that even though the total SLOC has increased by 50% that all the functionality has not been fully implemented. In fact even if we know that the contractor has reallocated 20% of the planned for functionality to another software module, which means the CPI should actually be approximately .53, this cannot be directly determined by using SLOC alone. Even in the case where we use a percentage of SLOC to determine earned value, there is still a possibility the EVM will overstate the progress made on the program.

Coding for all software modules is not completed at the same time. This provides an opportunity to compare planned SLOC with actual SLOC for modules developed early in the effort. This comparison provides a means to determine the accuracy of the initial estimate and can then be used to further refine the size estimate of code yet to be implemented. This is roughly similar to how EVM is used to estimate the final costs or EAC based on cost and schedule variances. If modules developed to date have experienced a 25% increase in SLOC in order to implement planned functionality, the same growth will probably be experienced for the remainder of the program.

3.2.1.3 SLOC EVM Issues Summary

Following are issues to consider when using a SLOC based EVM approach:

1. Definition of SLOC. Since SLOC can be defined in many ways, there must be agreement on the counting methods and rules used to determine total lines of code.

2. SLOC Counts. The estimated SLOC must be continuously updated throughout the software development life cycle, as data becomes available.

3. Effective Use of SLOC. SLOC is only appropriate for use during the code and unit test phase. Even in the code & unit test phase, if used as the basis of determining BCWP it is likely to yield very inaccurate results.

4. SLOC Growth. Initial SLOC estimates are often low, this means if they are used, as the basis for taking earned value, earned value is likely to overestimate the progress made.

5. SLOC & Requirements. SLOC does not directly relate to functionality implemented. If all the planned functionality is not implemented, earned value based on SLOC will overestimate the progress made.

Refer to the following for additional information:
1. Executive Summary
2.7 Software Code Issues
3.1.3.3 Code & Unit Test (C&UT) Phase (Requirements Metric)
3.2.1 Code & Unit Test Phase (SLOC Metric)
3.2.3.3.3 Code & Unit Test Phase (Function Points Metrics)

3.2.2 Equivalent SLOC (ESLOC)

3.2.2.1 Recommendation

ESLOC as an EVM Measure - Poor Not recommended as a basis for allocating BCWP

ESLOC has all of the disadvantages of SLOC plus additional inaccuracies caused by attempting to merge together different development methods in order to get a single equivalent SLOC. ESLOC is not recommended as a method for allocating earned value.

3.2.2.2 Overview & Description

The purpose of ESLOC is to attempt to normalize the size of the development effort for the different types of code being utilized (new, generated, modified, reused, and deleted, etc.). The different types of SLOC implementation are normalized to the amount of new SLOC which would require the same amount of effort to implement. This normalization is not a precise process.

Most software estimation models (COCOMO (COnstructive COst MOdel) II, SEER-SEM, SLIM, PRICE-S, etc.) have their own formulas for ESLOC based on analysis of historical data from hundreds and often thousands of software development efforts. Many software development companies have their own formulas based on analysis of their previous software developments. Most of these formulas are considered to be proprietary or sensitive information.

These formulas are essentially a simplification used to make the estimation process easier. The formulas are most accurate when, based on actual accurate data on the effort and schedule required for the different types of code to be developed or implemented, is tracked separately from the other types. The data can then be used to refine and improve the accuracy of the ESLOC formula. If such data has not been tracked, the accuracy of any ESLOC formula is questionable. Even if an accurate formula is available, the effort to develop the different types of code should continue to be tracked individually so that as improved technology and processes enhances the development process it is possible to continuously verify the accuracy of the ESLOC formula. See Appendix G for a discussion of the COCOMO II ESLOC formula.

EXAMPLE:

For build A, 10K SLOC of new code is developed and 20K SLOC of unmodified code is reused. Using the COCOMO II ESLOC equation, the level of effort to integrate 20K SLOC of reused code into the system is:

DM = 0, No redesign of reused code.
CM = 0, No design of reused code.
IM = 10, Integration effort for reused code.
AA = 0, code is well documented.

UNFM = SU = 0, UNFM and SU are set to 0 when code is reused unmodified.

$$AAF = (0.4 \times 0) + (0.3 \times 0) + (0.3 \times 10) = 3$$

$$\textbf{AAM} = \left[\frac{0 + 3 \times (1 + (0.02 \times 0 \times 0))}{100} \right] = .03$$

$$\text{Equivalent SLOC} = 20{,}000 \times (1 - {}^{0}\!/_{100}) \times .03 = 600$$

The 20K SLOC reused code is equivalent to 600 SLOC new code, or 10,600 ESLOC for new and reused code.

Assumption: The developer has historical data showing they are able to develop 4.5 SLOC or ESLOC per day for all development phases and all labor categories. One would expect it to require 2,356 staff days to develop build A.

Scenario: The reused and new code is not differentiated and is instead tracked together. In this case, at the end of build A, we have 30K SLOC which took 2,356 staff days to produce, which results in a productivity of 12.7 SLOC per day.

Assumption: Build B will consist of 20K SLOC new code and 10K SLOC unmodified reused code. This is equivalent to: 20,300 ESLOC using the COCOMO II ESLOC equations with the same DM, CM, IM, AA and UNFM as for build A. Once more assuming the developer can produce 4.5 ESLOC a day, build B would require 4,511 staff days.

Scenario: The reused and new code is not tracked separately. In this case, the choice may be made to revise the amount of staff hours required for build B based on the performance experienced in build A, 12.7 SLOC per day. In this case the number of staff hours required for build B would be 30,000 SLOC divided by 12.7 SLOC per day for 2362 staff days. However, since build A and build B have much different breakdowns of new and reused code, using the 12.7 SLOC per day estimate results in a much lower number of staff hours than will actually be required. The previous estimate of 4511 staff days is much more realistic since it adjusts for the much higher percentage of new code in Build B.

3.2.2.3 ESLOC EVM Issues Summary

EVM issues for ESLOC are very similar to those previously discussed for SLOC. It is possible for the total number of SLOC to stay constant, but have the amount of new code increase while reused and modified code decreases, thus resulting in an increase in project time and schedule. In this same situation, the amount of ESLOC would increase. ESLOC is not recommended as a method for allocating earned value.

Refer to the following for additional information:

1. Executive Summary

2.7 Software Code Issues
3.1.3.3 Code & Unit Test (C&UT) Phase (Requirements Metric)
3.2.1 Code & Unit Test Phase (SLOC Metric)
3.2.3.3.3 Code & Unit Test Phase (Function Points Metrics)

3.2.3 Function Points (FP)

3.2.3.1 Recommendation

Function Points as an EVM Measure - Poor Not recommended as a basis for allocating BCWP

If function points are used as the basis for determining BCWP it is essential that the FP count be continuously updated based on current system requirements. If this is not done, BCWP accuracy will decline similar to what happens when SLOC is used. Since FPs are applicable to most development phases they are likely to be more useful than SLOC for earned value purposes. However, if FPs are used as the basis for determining BCWP, the software requirements planned for implementation must still be tracked to insure they actually are implemented. Even though FPs appear to be a superior measurement to SLOC for use with EVM, they are only used rarely in software developed for NAVAIR. Most NAVAIR software developers use SLOC for size estimation. FPs remain inferior to requirements for tracking earned value.

3.2.3.2 Overview & Description

Function Points provide an alternative method to calculate software size. The size estimate is based on what the system does and as the systems functionality increases the number of function points increase. A function point could be thought of as a standard unit of software functionality. Function Points are directly derived from software requirements using a rigorously defined set of counting rules. The International Function Point Users Group (IFPUG)[1] function point counting rules are the most widely recognized standard. However, there are several other variations and derivatives for counting function points.

Additionally, where different requirements may take different amounts of effort to implement, each function point should take the same amount of effort to implement. This assumes that "*ADJUSTED*" function points are used. Adjusted function points take into account the domain in which the software will operate to adjust the function point count for system complexity. The unadjusted or "*RAW*" function point count for signal processing and web page software may be the same, but the adjusted function point count for the signal processing software will be higher because of its more complex domain. This simplifies determining earned value based on FPs in comparison to requirements.

A function point count requires expertise in applying the function point counting rules rather than the domain expertise necessary to do an accurate SLOC estimate. Detailed software requirements are required to accurately count either function points or SLOC. ***WHEN PERFORMING A FP COUNT, AT LEAST ONE OF THE TEAM MEMBERS MUST BE A CERTIFIED***

FUNCTION POINT SPECIALIST (CFPS) AND ALL TEAM MEMBERS SHOULD HAVE HAD A FUNCTION POINT COUNTING COURSE. In order to become a CFPS, a function point counter must pass a test administered by IFPUG. A CFPS can count approximately 1000 FPs a day, the equivalent of approximately 55 KSLOC of C++ code[2]. Ensuring that a CFPS leads the team results in a high level of consistency and accuracy in the size estimate, within plus or minus 10% for different CFPS[3]. Failure to used trained and certified function point counters looses most of the advantages a function point count has over a SLOC count in that the accuracy and rigor of the methodology is degraded. There is no reason that a similar degree of accuracy cannot be achieved in a SLOC size estimate as for Function Points if a rigorous and disciplined SLOC estimation method is used. However, many organizations have an inadequately disciplined SLOC estimation methodology, which often results in SLOC size estimates that are so too low.

3.2.3.3 Phases Using Function Points for EVM

3.2.3.3.1 Software Requirements Analysis Phase

Since the purpose of the Software Requirements Analysis phase is to decompose the systems requirements allocated to software into software requirements, it is difficult to use function points as the basis of taking earned value in this phase. This is due to the difficulty of generating an accurate function point count prior to the completion of the software requirements analysis phase. Any function point counts done prior to this point would have had to be based primarily upon analogy or preliminary (immature) systems requirements allocation and software architecture. This significantly reduces the accuracy of the function point count. This said, this preliminary function point count might very well be the best available information upon which to base the size of the software requirements analysis effort. **BECAUSE ANY FP COUNTS AVAILABLE AT THIS POINT ARE LIKELY TO BE VERY INACCURATE, EARNED VALUE SHOULD BE DONE BASED UPON THE PERCENTAGE OF FPS COMPLETED, VICE A FIXED NUMBER OF FPS.** A logical point at which to consider the requirements analysis complete is when it has **SUCCESSFULLY completed** a requirements peer review with all noted defects corrected.

EXAMPLE:

Assumptions: 1) Software requirements analysis is performed for 1000 FPs which equates to 10 system requirements. 2) An equal number of FPs are required for each system requirement. 3) BAC = $1M, 10-month schedule, BCWS = $100K each month.
Status: In month 5, the current FP estimate has increased to 1100 FPs and software requirements analysis has been completed on 500 FPs.

If percentage of completed FPs is the basis for EVM, what is the earned value at this point?
BCWS = $500K, BCWP = $454. 5K, ACWP = $500K
CPI = .91, SPI = .91

If earned value were based on the number of FPs completed vice the percentage:
BCWS = $500K, BCWP = $500K, ACWP = $500K
CPI = 1, SPI = 1

Obviously this does not accurately reflect the actual state of the program. However this situation cannot be corrected without rebaselining using the higher FP counts. This could require several rebaselinings as the count was corrected which is highly undesirable.

Status: In month 10, the current FP estimate has increased to 1200 FPs and software requirements analysis has been completed on 1000 FPs.

Using the percentage of total FPs as the basis for earned value:
BCWS = $1M, BCWP = $833K, ACWP = $1M
CPI = .833, SPI = .833

Assuming the same productivity of 100 FPs per month is maintained, at the 12-month point:
BCWS = $1M, BCWP = $1M, ACWP = $1.2M
CPI = .833, SPI = 1.0

This example assumes that all of the systems requirements allocated to be decomposed into software requirements during this phase actually were completed. If this is not the case, then the CPI and SPI will be indicating more progress than was actually made. If on the other hand,
only 8 of the 10 systems requirements had been decomposed, even after 12 months, then the following would be the case:
BCWS = $1M, BCWP = $800K, ACWP = $1.2M
CPI = .666, SPI = .8

Assuming it takes the same number of FPs to implement each systems requirement.

Because accurately counting FPs at the beginning of this phase is impractical, this can result in significant deviations from the expected cost and schedule. This can then ripple through the remainder of the program causing cost and schedule increases due to the increased effort required to complete the larger effort determined to actually exist at the end of the software requirements analysis phase. If such a situation actually occurs, the program should be restructured since it is obviously not going to meet its original cost and schedule objectives.

Refer to the following for additional information:
1. Executive Summary
3.1.3.1 Software Requirements Analysis Phase (EVM & Requirements Metric)
3.2.3.3.1 Software Requirements Analysis Phase (EVM & Function Points Metric)
3.1.4 Deferred Functionality or Requirements
3.1.5 Capacity & Performance Requirements Issues
3.1.6 General Requirements Issues

3.2.3.3.2 Software Design Phase

During the software design phase the software requirements are further developed into a software architecture and design from which the software source code can be directly developed. By the start of the design phase, an accurate function point count should be possible based on the software requirements defined in the software requirements analysis phase. This does not mean that further requirements changes may not occur as a result of the developers understanding of the system improving as development progresses. If such software requirements changes occur, the FP count must be updated to take into account the changes in requirements. Such changes should not have a contractual impact. If the changes are a result of Government changes in systems requirements, then they will require an Engineering Change Proposal (ECP) that will require a rebaselining of the effort to reflect the ECP.

Because the possibility of changing the number of function points to be implemented during design exists, earned value should be taken based upon a percentage of the current function point count for the design phase. Because a function point count must be done manually, the only way to determine how many function points have been designed is by counting the number of function points for the requirements that have been designed. A logical point at which to consider the software design completed is when it has **SUCCESSFULLY COMPLETED** a software design peer review with all noted defects corrected.

EXAMPLE:

Assumptions: 1) Design for 1000 FPs will be developed in the design phase of release A. 2) The developer's historical data indicates that they will require $450K (BAC) and 5 weeks to develop the design and peer review. 3) 200 FPs will be designed each week.
Status: At the end of week two, the FP count has increased to 1010 due to requirements Changes, 410 FPs have been implemented at a cost of $182.7K

After two weeks we planned to have 40% of the function points designed, in actuality we have 40.6% designed.

BCWS = $180K, BCWP = $182.7K, ACWP = $182.7K
CPI = 1.0, SPI = 1.015

After 5 weeks, the number of function points is 1025, and all 1025 have been implemented at the planned cost.

BCWS = $450K, BCWP = $450K, ACWP = $450K
CPI = 1.0, SPI = 1.0

Even though the number of function points increased from the original estimate of 1000, the amount of earned value earned for completing all of them did not. In this case the increase in FPs was due to software requirements changes resulting from an improved understanding of the software requirements. Since the FP increase was not caused by a change in the Government's requirements the planned cost to design the requirements, or BCWS for the design task, should not be impacted. In this case it appears the developer was very good at estimating exactly how much growth in requirements were likely to occur during the design phase and taking it into account when developing their project plan.

What would have happened if some of these changes were caused by changes to the Government's system requirements, resulting in a change in the software requirements to be designed? In that case, an ECP would have been required. Also, the later a requirements change is made, the higher the cost of that change. When a requirement is deleted late in the development, the design, coding and some of the testing may have been completed. Once it is deleted the design and code must be revised to remove the requirements functionality, retested and any defects resulting from the deletion corrected. In fact, deletion of requirements late in the development may actually be more costly in some cases than completing them.

Refer to the following for additional information:
1. Executive Summary
3.1.3.2 Software Design (Requirements Metric)
3.3 Modules (EVM & Modules Metric)

3.2.3.3.3 Code & Unit Test Phase

The differences between using function points as the basis for earned value during the code and unit test phase and during the design phase are slight. Earned value should be allocated based on the percentage of the most recent function point count for the software requirements to be implemented as part of the code and unit test phase. Again, changes in the number of function points can occur during the phase as the software requirements are further refined or if the customer changes their requirements for the system. **THE LOGICAL POINT TO DETERMINE WHEN A UNIT OF CODE HAS BEEN COMPLETED, AND EARNED VALUE CAN BE TAKEN FOR THE NUMBER OF FUNCTION POINTS IT IMPLEMENTS IS WHEN IT HAS SUCCESSFULLY COMPLETED A PEER REVIEW AND UNIT TEST WITH ALL DEFECTS CORRECTED.**

EXAMPLE:

C&UT Schedule: 9 months, 111 FPs planned for completion each month.
Assumptions: 1) C&UT must be performed for 1000 FPs 2) BCWS = $600K per month or $5.4M for the entire effort.
Status: At the end of 4 months, C&UT for 450 FPs has been completed along with Peer Review and all noted defects corrected at accost of $2.55M.. What is the earned value at this point?

$$BCWP = (\$5.4M) \times 490 / 1000 = \$2.646M$$

BCWS = $2.4M
ACWP = $2.55M

$$CPI = BCWP / ACWP = \$2.646M / \$2.55M = 1.038$$

$$SPI = BCWP / BCWS = \$2.646M / \$2.4M = 1.103$$

Status: At the end of 7 months, C&UT for 850 FPs has been completed along with Peer Review and all noted defects corrected. However, a series of changes to software and systems requirements have resulted in an increase in the number of software requirements to 1090 and made changes and deletions which reduced the number of FPs

for which C&UT was complete to 750. 110 FPs are changes due to Government changes. Schedule has been increased to 10 months and total BCWS for the effort to $6M. BCWS per month remains $600K. This schedule and funding increase is only for C&UT, not other development phases. What is the earned value at this point?

$$BCWP = (\$6M) \times 750/1090 = \$4.128M$$

BCWS = $4.2M

ACWP = $4.67M

$$CPI = BCWP/ACWP = \$4.128M/\$4.67M = .884$$

$$SPI = BCWP/BCWS = \$4.128M/\$4.2M = .983$$

Status: At the end of 10 months, C&UT for the total 1090 FPs has been completed along with Peer Review and all noted defects corrected. What is the earned value at this point?

BCWP = $6M

BCWS = $6M

ACWP = $6.538M

$$CPI = BCWP/ACWP = \$6M/\$6.538M = .918$$

$$SPI = BCWP/BCWS = \$6M/\$6M = 1.0$$

Refer to the following for additional information:

1. Executive Summary
2.7 Software Code Issues
3.1.3.3 Code & Unit Test (C&UT) Phase (Requirements Metric)
3.2.1 Code & Unit Test Phase (SLOC Metric)

3.2.3.3.4 Test Phases

Earned value should be allocated during testing phases when the software requirements have been successfully tested. Once the **SUCCESSFUL** test has occurred, the earned value for the number of FPs for the requirements tested can be taken. Remember, running a test is not the same as successfully completing the test. Earned value can't be taken until the requirements have been successfully tested.

EXAMPLE: *Determining Progress In Test Plan and Test Procedure Development*

Schedule: 21 months, 1000 FPs need test procedures developed.

Assumptions: BCWS = $.152M per month and $3.192M for entire effort.

Status: At the end of 10 months, test procedures and peer review for 475 FPs are completed at a cost of $1.5M.

What is the earned value at this point?

$$BCWP = (\$3.192M) \times 475/1000 = \$1.516M$$

BCWS = $1.52M

ACWP = $1.5M

$$CPI = BCWP/ACWP = \$1.516M/\$1.5M = 1.011$$

$$SPI = BCWP/BCWS = \$1.516M/\$1.52M = .997$$

Status: At the end of 20 months, test procedures and peer review for 950 FPs have been completed, however the total number of FPs has increased to 1400. What is the earned value at this point?

$$BCWP = (\$3.192M) \times 950/1400 = \$2.166M$$

BCWS = $3.04M

ACWP = $3M

$$CPI = BCWP/ACWP = \$2.166M/\$3M = .722$$

$$SPI = BCWP/BCWS = \$2.166M/\$3.04M = .7125$$

Status: At the end of 29 months, test procedures and peer review for all 1400 have been Completed. What is the earned value at this point?

BCWP = $3.192M

BCWS = $3.192M

ACWP = $4.421M

$$CPI = BCWP/ACWP = \$3.192M/\$4.421M = .722$$

$$SPI = BCWP/BCWS = \$3.192M/\$3.192M = 1.0$$

Obviously this task experienced a large overrun on schedule and cost. It may further have delayed the start and/or completion of testing due to the procedures not being ready. Delay in testing would depend on how much overlap there was in the test procedure development and test schedules. The more overlap, the more likely that there would be a delay in testing.

EXAMPLE: *Determining Progress In Test Completion*

Scenario: Software integration testing conducted on 1000 FPs, 6-month schedule, BCWS = $1.52M per month or $9.12M for the total task. Test phase includes not only personnel and resources for conducting the test, but also personnel and resources for correcting defects found during testing and the reexecution of appropriate test procedures. The project plan calls for 95% of the 1000 FPs to be successfully tested; the remainder will be deferred for correction to a later build. Some of the code must pass testing in order to avoid critical path

impacts on later builds.

Assumptions: Only non-critical path code is deferred. Quality requirements for software – No more than 10 priority 3 defects when code delivered or one priority 3 defects per 100 FPs. This means that there can be no more than 9 uncorrected priority 3 defects in 950 FPs which pass testing. All priority 1 and 2 defects must be corrected.

Status: At the end of 3 months of testing, test procedures have been performed for 610 FPs. There are two priority 2 defects and 10 priority 3 defects currently open. The two priority 2 defects must be corrected along with at least 4 of the priority 3 defects. 485 of the tested FPs are considered to have been successfully tested since they are not affected by the priority 2 defects and have less than 1 priority 3 defects per 100 FPs. What is the earned value at this point?

$$BCWP = (\$9.12M) \times 485/950 = \$4.656M$$

BCWS = $4.56M

ACWP = $4.51M

$$CPI = BCWP/ACWP = \$4.656M/\$4.51M = 1.032$$

$$SPI = BCWP/BCWS = \$4.656M/\$4.56M = 1.021$$

Status: At the end of 6 months of testing, test procedures on all of the FPs have been completed. There is one priority 1 defect open and 11 priority 3 defects. 60 FPs of software containing the priority 1 defect and 2 of the priority 3 defects have been deferred to a later build. What is the earned value at this point?

$$BCWP = (\$9.12M) \times 940/950 = \$9.024M$$

BCWS = $9.12M

ACWP = $9.16M

$$CPI = BCWP/ACWP = \$9.024M/\$9.16M = .985$$

$$SPI = BCWP/BCWS = \$9.024M/\$9.12M = .989$$

Since more than 5% or 50 FPs were deferred to a later build, all of the BCWS cannot be earned at this time. The remainder will not be earned until the extra 10 deferred FPs are successfully tested in a later build.

As can be seen by the previous example, quality requirements can have a significant impact on the determination of earned value. Determining how many FPs are affected by defects increases the level of traceability needed by the program, thus increasing costs. A defect is more naturally and easily traced to the specific software requirements it impacts. The software requirement must be than traced to the number of FPs of code that implements it. This step can be avoided if software requirements are instead used as the basis for allocating earned value. Determining the number of FPs affected by a defect can also be rather subjective with a tendency to minimize the

number of FPs in order to improve the earned value numbers. This can result in reduced accuracy. Do not replan if the amount of differed functionality exceeds the project plan. Replanning will negate earned values ability to indicate cost and schedule slips caused by excessive functionality deferral.

Refer to the following for additional information:
1. Executive Summary
3.1.3.4 Test Phases (EVM & Requirements Metric)
3.4 Test Procedures/Cases (EVM & Test procedures/Cases Metric)

3.2.3.3.5 Rework

As with requirements, while it may be possible to predict the number of function points worth of requirements which will fail testing and require rework. It is very difficult to determine how much time to allocate for rework based upon this prediction. Time and effort for rework is usually based on the developers estimate of the number of defects likely to occur and the average amount of time required to correct such defects.

Refer to the following for additional information:
1. Executive Summary
3.1.3.5 Software Rework (EVM & Requirements Metric)
3.5 Software Defects (EVM & Software Defects Metric)

3.2.3.3.6 Capacity, Performance and General Requirements Issues

The issues associated with capacity, performance and general requirements are the same as defined in sections 3.1.5 and 3.1.6 for requirements. Function points simply provide a means of determining the amount of effort necessary to implement different requirements.

3.2.3.4 FP EVM Issues Summary

Following are issues to consider when using a FP based EVM approach:

1. Trained personnel should perform FP counting. At least one of the team members must be a Certified Function Point Specialist (CFPS).

2. FP counts are best performed on adequately defined software requirements. Requirements should be specified at the level of detail found in a Software Requirements Specification prior to attempting to perform a function point count. Other methods such as analogy will be used prior to this point.

3. FP counts must be continually updated to reflect changes in software requirements. Unforeseen slips in the critical path can result even though the number of completed function points indicates that the project is ahead of schedule. Failure to update the FP counts when requirements change negates most of the advantages of FPs as a sizing tool.

4. Using FPs to determine EVM is applicable to all phases of software development. There may be specific tasks in each phase, which are not well suited to measuring EVM using FPs.

5. Unlike SLOC, FPs cannot be automatically counted (currently). Thus the only way to know how many FPs of work have been completed in each phase is by using a trained team of functions point counters led by a CFPS to insure that the current count reflects any changes to the software requirements. Additionally, the number of FPs required to implement a requirement(s) must be tracked to the requirement(s) so that it is possible to determine how much earned value should be allocated for the completion of that requirement(s) in each development phase. This is not any different than what should be done with SLOC in order to maintain an accurate estimate, except when the code is actually finished an automatic counting tool can be executed in order to get the actual final SLOC count.

6. Keep in mind that at this time FPs are only rarely used by NAVAIR software developers, either in industry or in house.

7. FPs are recommended with reservations for allocating BCWP.

3.3 Modules

3.3.1 Recommendation

Modules as an EVM Measure – Poor Not recommended as a basis for allocating BCWP

Modules share many of the same disadvantages as SLOC in that they are often not directly correlated to the implementation of software requirements. Modules are often declared finished even though they do not implement all of the planned requirements in order to preserve the illusion of being on schedule. If the completion of modules is used for determining BCWP in this case, the results will be unrealistically high values for CPI and SPI. Thus if modules are used, software requirements implementation must also be tracked in order to insure the modules implement all of the requirements they were planned to prior to earning all of the associated BCWP. This makes them a poor metric for allocating BCWP.

3.3.2 Overview & Description

The term modules will be used to collectively refer to Computer Software Components (CSCs), Computer Software Units (CSUs), classes, packages or other similar units of code below the Computer Software Configuration Item (CSCI) level. CSCIs are divided into CSCs and CSUs that represent the lowest sub-function of the software. Refer to Figure 3-1.

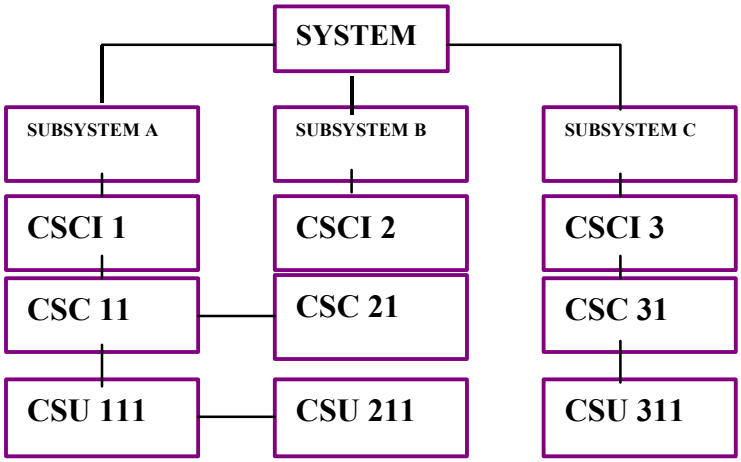


Figure 3-1: System Hierarchy for Software Development

When using CSC/CSUs as the basis for allocating earned value, earned value would be earned when a specific task in the CSC/CSUs development had been completed. Thus when the design of the CSC/CSU was complete, earned value for the design phase would be earned, when code and unit test for the CSC/CSU was complete, earned value for the code and unit test phase would be earned. For the design and code and unit test phases, completion could be determined by the successful completion of the peer reviews for each phase. During CSCI or software integration testing, earned value would be awarded based on successful completion of the CSC/CSUs testing.

When using SLOC for earned value purposes it is assumed that each line of SLOC requires the same amount of effort, this assumption is not necessarily the case for CSC/CSUs. Different modules can include varying degrees of functionality and the effort to design and code them can vary widely. Unless the developer makes a conscious decision as part of their development effort to make all CSC and CSUs the same size, assuming that all CSCs or CSUs will require a similar amount of effort is a risky assumption. In the case where CSCs and CSUs vary in the amount of functionality they implement, it will be necessary to individually estimate the amount of effort required to implement each, based on the requirements traced to them and/or their size estimates.

Earned value should be allocated for CSC/CSUs upon completion of each development phase (design, code and unit test, software integration test) for the module, essentially using the milestone method, where no earned value is earned until the milestone is achieved. The full-earned value, or BCWS, for the phase should also only be allocated if all of the software requirements or design planned for implementation in the CSC/CSU was actually included. If a CSC/CSU includes only 50% of its designed functionality, only 50% of the module's BCWS can be allocated, even if the ACWP exceeds the BCWS. The remainder of the BCWS for the module cannot be earned until the missing requirements or design are implemented in either the CSC/CSU in question or some other CSC/CSU.

Attempting to allocate earned value during software integration testing based on CSUs may also give an inaccurate picture of progress. Many test procedures may be executed successfully, but because they only partially exercise the functionality in several CSC/CSUs, no earned value will be awarded based on completing testing for the CSC/CSU. Thus during initial testing, very little earned value may be awarded, while later in the testing phase earned value is earned very quickly as the functionality of the various CSC/CSUs finally complete testing. In this case earned value may indicate less progress being made than is actually the case, especially if this is not considered in laying out work packages and cost accounts for the testing phase.

As with other non-requirements methods, allocating BCWP based on completion of modules does not provide direct insight into which or how many software requirements have been implemented. Requirements must still be tracked in order to verify what has been implemented in comparison to the plan and to insure modules are not allocated all the planned BCWP if they don't include all the planned requirements. Allocating BCWP based on Modules is thus more complicated than basing it on requirements since it adds extra non-value added steps to the process.

3.3.3 Modules EVM Issues Summary

1. Use of module completion alone as a basis for allocating BCWP will not guarantee all of the requirements planned for the module are completed. Modules have often been declared as completed even though all requirements are not implemented. Thus requirements must also be tracked to verify their implementation.

2. Can be used as basis for allocating BCWP in design, code & unit test and software integration testing as long as it is also verified that the planned requirements for the module have actually been implemented in each of these phases.

3.4 Test Procedures/Cases

3.4.1 Recommendation

Test Procedures/Cases as an EVM Measure – Good Highly recommended as a basis for allocating BCWP

If each individual test procedure/case identifies the software or systems requirements that it is intended to verify, using the "successful" completion of a test procedure is a good method for determining when to allocate BCWP during test phases. Successfully passing the test procedure/case will also identify what requirements have been successfully implemented. If more than one test procedure/case is required to verify the requirement in question, all of the test procedures must be successfully completed. It is also important to insure that requirements are fully verified by one or more test procedures/cases.

3.4.2 Overview & Description

Test plans and procedures are intended to verify the correct implementation of system and software requirements. Test plans must include adequate time and resources to allow for re-

testing of any failed test procedures. This also includes rework of requirements, design and code to correct defects found in testing. This requires the developer to estimate the likely amount of rework time for different builds and test phases based on historical data.

Historical data from previous development efforts can be used to estimate the average number of test cases. Typically, unit test cases are included with coding and both are considered complete once a peer review of the code has been completed and unit test cases have been completed. Unit test procedures should not be used as a method allocating BCWP and are not considered as part of this discussion.

BCWP should be allocated only for the test procedures/cases that have been successfully completed. Including the number of failed test procedures/cases provides no useful information and results in the earned value indicating more progress has been made than is actually the case. Successful completion of a test procedure/case is determined by the software contractual quality requirements. At a minimum this means that if any priority 1 or 2 defects occur during the test procedure/case, the test has been unsuccessful[9]. How many priority 3, 4 and 5 defects are permissible are determined by contractual quality requirements. As long as the number of and priority of defects occurring in a test phase are low enough, so that when summed with the uncorrected defects from other test phases are within program quality requirements, the test procedures/cases may still be considered to have passed and the associated requirements to have been adequately implemented. This requires the developer to estimate what an acceptable number of priority 3, 4 and 5 defects are for earlier test phases that will keep the final totals of open defects within program quality requirements. This suggests that the developer strive to correct as many defects as possible as early as possible in order to avoid the build up of a huge bow wave of defects that must be corrected prior to entering OPEVAL.

As part of the development of test plans and procedures, it must be verified that all of the software requirements are tested. As with all other EVM methods the primary objective is to determine the progress being made in implementing the Governments requirements. If test procedures do not test all of the software requirements, EVM will provide an incomplete and inaccurate view of test progress.

3.4.3 Test Procedures/Cases EVM Issues Summary

1. ***PROGRAM QUALITY REQUIREMENTS MUST BE CONTRACTUALLY SPECIFIED IN ORDER TO MINIMIZE SUBJECTIVITY IN DETERMINING IF A PARTICULAR DEFECT HAS CAUSED THE TEST PROCEDURE/CASE TO FAIL OR NOT. THIS IS NECESSARY NO MATTER WHAT METHOD IS USED TO ALLOCATE EARNED VALUE.***

2. Applicable only to non-code and unit test, test phases.

3. Each test procedure/case must be traced to the system/software requirement it verifies in order to determine which requirements have been correctly implemented. This should be done in any case by any organization which has achieved a SW-CMM® or CMMI® level 3.

Refer to the following for additional information:

3.1.3.4 Test Phases (EVM & Requirements Metric)
3.2.3.3.4 Test Phases (EVM & Function Points Metric)

3.5 Software Defects

3.5.1 Recommendation

Software Defects as an EVM Measure – Poor Not recommended as a basis for allocating BCWP

During the planning phase the number of defects expected to occur is probably based on the size of the effort. Thus defect estimates are twice removed from requirements, this makes the correlation to actual requirements implementation for earned value purposes even weaker than that for SLOC. In some cases several defects may need to be corrected before a requirement is considered to be correctly implemented, in other cases correcting a single defect can result in several requirements being correctly implemented. This further complicates allocating BCWP based on defect correction. Defect correction will occur during rework phases along with subsequent requirements analysis, design and code & unit test phases in later builds. Using defects to allocate BCWP in these non-rework phases is even more difficult than in using them in pure rework phases and should be avoided. **WHILE DEFECT ESTIMATES ARE USEFUL FOR PLANNING PURPOSES IN ALLOCATING RESOURCES AND TIME TO CORRECT DEFECTS, THEY ARE UNSUITABLE FOR USE AS THE BASIS OF ALLOCATING BCWP AND SHOULD BE AVOIDED.**

3.5.2 Overview & Description

For efforts in excess of 10,000 function points, (55,000 KSLOC C++), the effort required for defect removal/repair can easily exceed twice that for the initial coding[4]. Time and resources must be allocated in the project schedule for rework of requirements, design, and coding to correct defects. Historical data on error rates from previous software development efforts should be used as a basis for predicting resources required for defect removal.

For the purpose of this discussion, only defects detected following the code & unit test (C&UT) phase are considered. Defects discovered during peer reviews or via unit test are corrected as part of the development process and should be avoided as an EVM measure.

Attempting to take earned value based on the correction of defects is difficult due to the wide variance of time and effort required to fix different defects. Some defects can be fixed easily in a few minutes; others will take hundreds of hours of analysis, requirements rework, design and code corrections followed by extensive regression testing. If the program is large, with large numbers of defects predicted, using an average amount of time and/or resources, as the basis of estimating rework resource and schedule needs may be an adequate method. For smaller projects, with fewer defects, this becomes increasingly inaccurate.

Programs may establish different difficulty levels for defects. The more difficult the defect the more time and/or resources required to correct the problem. The program may then attempt to predict the numbers of the different difficulty levels of defects that will occur and the amount of

time and resources needed to correct them. By grouping defects by difficulty, the variance in effort to correct them can be reduced.

EXAMPLE:

Effort: 100 defects are predicted to occur during software integration testing of Build 1. Based on historical data, it will take on average 50 staff hours to correct each defect. Correcting includes any rework of requirements, design, code and regression testing necessary to repair the defect. Assume the contractors standard labor rate is $100 per staff hour. Further assume that the project plan calls for correcting 20 defects per month, or 1000 hours of defect correction per month, or $100,000 BCWS per month. See Table 3-1 for additional project EVM data. All defects in the example need to be corrected to meet program quality requirements.

Status: <u>At the end of month 1:</u>
BCWS = $100,000
ACWP = $75,000
BCWP = $75,000
CPI = 1, SPI = .75

What does this mean? It depends:

1. What if only 15 defects had been detected? Remember the 20 defects per month was a prediction. If it was too high, as in this case, it makes the effort look as if it is behind schedule, however the defect correction test has corrected all available defects.

2. The first reaction from hearing that there are fewer defects than predicted is that this is a good thing. Not necessarily, until we know why this is the case. What if Build 1 software integration testing is running behind schedule? This could account for the low number of defects and would further justify the low SPI. If on the other hand software integration testing is on schedule, the SPI is falsely indicating the rework effort is running behind.

3. What if 30 defects were outstanding during the first month as a result of the Build 1 software integration testing? Thus there are actually 50% more defects to correct than predicted. If testing is on schedule, this means that the SPI is too high in comparison to actual progress. On the other hand, what if testing was ahead of schedule? If testing was ahead of schedule by about 50%, this could account for the additional defects. However, even if this is the case, defect correction is still behind. But how far behind? Should it be considered to be behind by 25%, since only 15 of the planned 20 defects have been corrected, or should it be considered to be 50% behind because it isn't keeping up with the accelerated testing?

For the remainder of the example assume that testing is on schedule but that 30 defects have been discovered to date vice the predicted 20.

Status: <u>At the end of month 3:</u>

$BCWS_{CUM}$ = $300,000	CPI_{CUM} = .81
BCWS = $100,000	CPI = .71
$BCWP_{CUM}$ = $300,000	SPI_{CUM} = 1.00
BCWP = $125,000	SPI = 1.25
$ACWP_{CUM}$ = $370,000	
ACWP = $175,000	

What does this mean?

1. EVM indicates the project is over budget but on or ahead of schedule. The project is definitely spending faster than predicted; costs per defect have risen from the original predicted $5000 per defect to $7000. What is causing this? Is it taking more time than expected to fix defects? Has the cost of the staff working on defects correction increased? Has the experience level, and thus the cost of the staff fixing defects increased?
2. By this time it was predicted that 60 defects would have been corrected. This is actually the case, however the total number of defects detected is 85. Thus while the correction level is keeping up with predictions, the number of defects that need to be corrected is greater than predicted. This would seem to indicate that the defect correction task is behind schedule.

Status: At the end of month 6:

$BCWS_{CUM}$ = $500,000	CPI_{CUM} = .75
BCWS = $0	CPI = .71
$BCWP_{CUM}$ = $675,000	SPI_{CUM} = 1.35
BCWP = $125,000	SPI =
$ACWP_{CUM}$ = $895,000	
ACWP = $175,000	

What does this mean?

1. Since the second month of the effort, EVM has shown the rework effort to be on or ahead of schedule. However, since over a third more defects were discovered than were predicted, the task ended up taking a month longer than predicted.
2. Obviously taking EVM in the method shown here does not accurately reflect the schedule situation. What could have been done to correct this?
3. Since the second month of the effort, EVM has shown the rework effort to be on or ahead of schedule. However, since over a third more defects were discovered than were predicted, the task ended up taking a month longer than predicted.
4. Obviously taking EVM in the method shown here does not accurately reflect the schedule situation. What could have been done to correct this?

Month	1	2	3	4	5	6
Predicted # of defects per month	20	20	20	20	20	0
Predicted Cumulative # of defects	20	40	60	80	100	100
Actual # of defects corrected per month	15	20	25	25	25	25
Actual cummulative # of corrected defects	15	35	60	85	110	135
Actual # of defects per month detected	30	30	25	25	25	0
Actual cumulative # of defects detected	30	60	85	110	135	135
Predicted $ per defect	$5,000	$5,000	$5,000	$5,000	$5,000	$5,000
Actual $ per defect	$5,000	$6,000	$7,000	$7,000	$7,000	$7,000
BCWS	$100,000	$100,000	$100,000	$100,000	$100,000	$0
BCWP	$75,000	$100,000	$125,000	$125,000	$125,000	$125,000
ACWP	$75,000	$120,000	$175,000	$175,000	$175,000	$175,000
$BCWS_{CUM}$	$100,000	$200,000	$300,000	$400,000	$500,000	$500,000
$BCWP_{CUM}$	$75,000	$175,000	$300,000	$425,000	$550,000	$675,000
$ACWP_{CUM}$	$75,000	$195,000	$370,000	$545,000	$720,000	$895,000
CPI	1	0.83	0.71	0.71	0.71	0.71
SPI	0.75	1.00	1.25	1.25	1.25	
CPI_{CUM}	1	0.90	0.81	0.78	0.76	0.75
SPI_{CUM}	0.75	0.88	1.00	1.06	1.10	1.35

Table 3-1: Defect Earned Value Example

As noted in the previous example, taking earned value based on defects is likely to cause problems if the number of defects predicted is different from what actually occurs. Similar problems were also noted as being a possibility for SLOC and FPs. As with SLOC and FPs, the problem can be at **LEAST PARTIALLY CORRECTED BY USING THE PERCENTAGE OF DEFECTS CORRECTED RATHER THAN A FIXED ESTIMATED VALUE**. Like with SLOC, using percentages of defects corrected can cause unusual side affects as the defect estimate changes such as BCWP reducing in subsequent months after a reestimation of defects.

Defects are assigned priorities depending on their impact on meeting systems requirements. The five different priorities are defined in IEEE/EIA 12207.2[5] and Table 3-3 below. Priority 1 and 2 defects must be corrected in order to commence OPEVAL[9]. In addition, all priority 3 defects must be documented with an impact analysis. Thus clearly, all priority 1 and 2 defects must be corrected as part of the development process. Priority 3 defects are more flexible. The program should have established some contractual quality criteria in which the maximum number of priority 3, 4 and 5 defects that could be open prior to acceptance and commencement of OPEVAL would be identified. While priority 3 defects don't necessarily have to be corrected prior to entering OPEVAL, if there are large numbers of them, the system is unlikely to pass OPEVAL. Priority 4 and 5 defects are not required to be corrected prior to OPEVAL. Thus which defects are required to be corrected are dependent on their priority.

Priority	Applies if a problem could
1	a) Prevent the accomplishment of an essential capability b) Jeopardize safety, security, or other requirement designated "critical"
2	a) Adversely affect the accomplishment of an essential capability and no work-around solution is known b) Adversely affect technical, cost, or schedule risks to the project or to life cycle support of the system, and no work-around solution is known
3	a) Adversely affect the accomplishment of an essential capability but a work-around solution is known b) Adversely affect technical, cost, or schedule risks to the project or to life cycle support of the system, but a work-around solution is known
4	a) Result in user/operator inconvenience or annoyance but does not affect a required operational or mission-essential capability b) Result in inconvenience or annoyance for development or maintenance personnel but does not prevent the accomplishment of the responsibilities of those personnel
5	Any other effect

Table 3-2: IEEE/EIA 12207.2 Defect Priorities

The predicted number of defects, and subsequently any BCWS for defect correction is generally based upon historical data and the number of defects generated per unit of size. If the size estimate for the effort is low, then the number of defects will also probably be low. **THUS ANY RE-ESTIMATION OF SOFTWARE SIZE SHOULD BE ACCOMPANIED BY A RE-ESTIMATION OF THE REWORK EFFORT REQUIRED TO CORRECT DEFECTS.**

In the discussion of the use of SLOC as a method of determining earned value, it was pointed out that the primary problem with using SLOC is that it is often underestimated and is not directly related to insuring that requirements are implemented. Defect estimates are often based on a SLOC, or some other size estimate, even further removing them from a direct correlation with requirements implementation. While metrics and measurements for tracking software defects are an essential part of tracking the status of and managing the software development, they are difficult to use in the determination of earned value.

3.5.3 Software Defects EVM Issues Summary

Software defects as a measure are only applicable to rework phases.

3.6 Schedule Milestones

3.6.1 Recommendation

Schedule Milestones as an EVM Measure – Poor Not recommended as the basis for allocating BCWP

Using Schedule Milestones is an extensively abused methodology where BCWP is allocated based on reaching some project milestone. Unfortunately the milestone is often declared to have been met even though all of the requirements of the milestone have not been achieved. This results in earned value giving an unrealistically positive view of the project status. Schedule Milestones are not recommended as a method for allocating BCWP.

3.6.2 Overview & Description

Builds, releases and other software schedule milestones are often considered completed without all of the planned requirements being accomplished. All of the earned value credit should not be taken at a milestone unless all of the requirements planned for that milestone have in fact been completed. If only a subset of the work is completed, only a subset of the earned value should be allocated. Requirements that have not been completed must then be implemented in a subsequent milestone.

EXAMPLE:

Estimated Effort: 1K requirements for the first build or release of a development
Status: Release or build delivered on time by the scheduled milestone date within budget
But contains only 750 requirements.

If schedule milestones is the basis of earned value, how much can be allocated to this effort?

Earned value would indicate that the milestone was met and 100% earned value would be allocated although 250 requirements have not been implemented. The remaining 250 requirements must be implemented at a later point in the development, which will contribute to a cost and schedule overrun of the total effort. To declare that schedule milestones have been met even though all requirements for the milestone have not been met reduces visibility into cost and schedule overruns.

3.6.3 Schedule and Milestone EVM Issues Summary

Following are issues to consider when using a milestone schedule based EVM approach:

Requirements Completion – Milestones are often declared complete even though all planned for functionality and requirements are not implemented. EVM systems must account for these incomplete requirements or it will indicate more progress than has actually occurred.

3.7 Level of Effort (LOE)

3.7.1 Recommendation

Level of Effort as an EVM Measure – Poor Not recommended as the basis for allocating BCWP

3.7.2 Overview & Description

By definition, LOE is work that does not result in a final or tangible end product. The basis of measurement is time so the earned value automatically starts to accumulate when the effort begins. LOE has no schedule variance so a meaningful schedule analysis cannot be performed, BCWP = BCWS. Theoretically, since LOE can generate a cost, ACWP can still be compared to the BCWP to get a meaningful result for variance analysis. However, because SPI will always be 1, anything the CPI tells us is automatically suspect. If CPI is less than one, it may mean labor rates have unexpectedly increased, or that the work is actually ahead of schedule, or it is taking a greater amount of hours to get done what was planned to be accomplished.

Level of effort should never be used for any task in which a product or artifact is being produced against which to measure program progress. In labor categories directly related to performing tasks such as: requirements analysis, design, code & unit test, software integration testing, systems testing, DT and OT, LOE should not be used. All of these areas are developing an intermediary product, or artifact leading to the completion of the project.

Level of effort may be more appropriate as a measure in indirect support development activities such as Management, Administrative Support, Software Configuration Management (SCM), Software Quality Assurance (SQA), Systems and Network Administration, Software Engineering Environment Administration and Support, Software Integration Test Environment Administration and support, etc.. . Even in these situations using LOE should be avoided if at all possible. It may be possible to take advantage of activity based costing systems when available to apportion these labor categories to specific tasks. The total management, SCM, SQA and other supporting tasks could be apportioned to tasks which directly contribute to the development of a requirements based software development artifact, such as: software requirements analysis, design, C&UT, test, etc..

EXAMPLE:

Effort: A program has identified Management, SQA, SCM and Software Engineering Environment Support tasks. These tasks while essential to the development do not directly produce any software products or deliverables; they will be referred to as support tasks.

Status: During the current month, $100K of BCWS is allocated to these tasks and the total BCWS for other tasks, which do produce software products and deliverables, are $1M, they will be referred to as direct tasks. Assume during the current month for software requirements analysis the BCWS is $500K or 50% of the direct task BCWS for

the month. This would mean that half of the BCWS for the support tasks is apportioned to the software requirements analysis task. Assume that at the end of the current month, the software requirements analysis tasks BCWP is $450K and its ACWP is $550K. For the other direct tasks, BCWP = $500K and ACWP = $500K. Assume for the support tasks the ACWP is $100K. This would result in the following:

SW requirements Analysis task CPI = .818
SW Requirements Analysis task SPI = .9
Other direct tasks CPI = 1.0
Other direct tasks SPI = 1.0

$$\text{Support Tasks } BCWP = [\$100K \times 50\% \times (\$450K/\$500K)] + [(\$100K \times 50\% \times (\$500K/\$500K)] = \$95K$$

Support Tasks CPI = .95
Support Tasks SPI = .95

Instead of using level of effort for the support tasks, BCWP is thus allocated based on the progress made in tasks that directly produce software development artifacts. If level of effort had been used, the CPI and SPI for the support tasks would have been 1.0. Since the tasks the support tasks were supporting did not perform according to the project plan, it is unreasonable to give the support tasks full credit for achieving the project plan.

[1] http://www.ifpug.org/
[2] Applied Software Measurement, Capers Jones, McGraw Hill, 1996, page 84.
[3] Curing the Software Requirements and Cost Estimating Blues, Mike Nelson, James Clark, Martha Spurlock, Program Manager Magazine, Nov/Dec 1999.
[4] Jones, T. Capers, Estimating Software Costs, McGraw-Hill, 1998
[5] IEEE/EIA 12207 The Software Life Cycle Process, page 94 – 96.

Intentionally left blank

APPENDIX A. ACRONYMS AND DEFINITIONS

ACAT **Acquisition Category**

ACWP **Actual Cost of Work Performed (Actual Cost)**
Costs actually incurred and recorded in accomplishing the work performed within a given time period. (Cost of work accomplished)

AE **Apportioned Effort**
Effort that by itself is not readily divisible into short-span work packages but which is related in direct proportion to measured effort.

BAC **Budget at Completion**
Sum of all budgets established for the contract.

BCWP **Budgeted Cost for Work Performed (Earned Value)**
Sum of the budgets for completed work packages and completed portions of open work packages, plus the applicable portion of the budgets for level of effort and apportioned effort. (Value of work accomplished)

BCWS **Budgeted Cost for Work Scheduled**
Sum of the budgets for all work packages, planning packages, etc., scheduled to be accomplished (including in-process work packages), plus the amount of level of effort and apportioned effort scheduled to be accomplished within a given time period.

C&UT **Code & Unit Test**

CFPS **Certified Function Point Specialist**

CMM® **Capability Maturity Model®**
A measure of software process maturity developed by the Software Engineering Institute (SEI) at Carnegie Mellon University.

CMMI® **Capability Maturity Model Integration®**

CMU **Carnegie Melon University**

COCOMO **COnstructive COst MOdel**

COTS **Commercial Off The Shelf**
May refer to either hardware or software and is typically purchased through a licensing agreement that may or may not include the source code, but, may require effort to maintain the configuration.

CPI **Cost Performance Index**
Performance index calculated as:
BCWP / ACWP

CPU **Computer Processing Unit or Central Processing Unit**

CSC/ CSCI/ CSU **Computer Software Component/ Computer Software Configuration Item/ Computer Software Unit**
A system is partitioned into various subsystems, which are further partitioned into CSCIs. A CSCI is a program, or group of programs, which satisfies a common end-use function and is managed separately. Since CSCIs may contain over 10,000 lines of code, they are further partitioned into CSCs and CSUs. CSUs are the lowest level software entities and usually contain between 100 and 200 lines of code.

DoD **Department of Defense**

DT **Developmental Test**

EAC **Estimate at Completion**
Actual direct costs, plus indirect costs allocable tot he contract, plus the estimate of costs (direct and indirect) for authorized work remaining.

ECP **Engineering Change Proposal**

EV **Earned Value**

EVM **Earned Value Management**

EVMS **Earned Value Management System**

FP **Function Points**

IBR **Integrated Baseline Review**

IFPUG **International Function Point Users Group**

IPT **Integrated Process Team**

KPP **Key Performance Parameter**

KSLOC **One Thousand Source Lines of Code**

LOE **Level of Effort**

MNS **Mission Needs Statement**
A formatted non-system specific statement containing operational capability needs and written in broad operational terms. It describes required operational capabilities and constraints to be studied during the Concept and Technology Development Phase.

MR **Management Reserve**

An amount of the total allocated budget withheld for management control purposes rather than designated for the accomplishment of a specific task or set of tasks. It is not part of the performance measurement baseline.

OPEVAL **Operational Evaluation**

ORD **Operations Requirements Document**
A formatted statement containing performance and related operational performance parameters for the proposed concept or system. Prepared by the user or user's representative at Milestone B and Milestone C.

OT **Operational Test**

PMB **Performance Measurement Baseline**
The time-phased budget plan against which contract performance is measured. It is formed by the budgets assigned to schedule cost accounts and the applicable indirect budgets. For future effort, not planned to the cost account level, the performance measurement baseline also includes budgets assigned to higher level CWBS elements, and undistributed budgets. It equals the total allocated budget less management reserve.

PSM **Practical Software Measurement/**
Practical Software and Systems Measurement

RAM **Random Access Memory**

SEI **Software Engineering Institute**

SLOC **Source Lines of Code**

SPI **Schedule Performance Index**
Performance index calculated as:
BCWP / BCWS

SRD **Software Requirements Description**
Per IEEE/EIA 12207

SRS **Systems Requirements Specification**
Per IEEE/EIA 12207

SW **Software**

SW-CMM® **Capability Maturity Model for Software**

TPM **Technical Performance Measure**

TRL **Technical Readiness Level**

WBS **Work Breakdown Structure**

A product-oriented family tree division of hardware, software, services, and other work tasks which organizes, defines, and graphically displays the product to be produced as well as the work to be accomplished to achieve the specified product.

WP **Work Package**

Detailed jobs, or material items, identified by the contractor in order to complete contractually required tasks. A work package has the following characteristics:

- It represents units of work at levels where work is performed.
- It is clearly distinguished from all other work packages
- It is assigned to a single organizational element.
- It has scheduled start and completion dates and, as applicable, interim milestones, all of which are representative of physical accomplishment.
- It has a budget or assigned value expressed in terms of dollars, man-hours, or other measurable units.
- Its duration is limited to a relatively short span of time or is subdivided by discrete value milestones to facilitate the objective measurement of work performed.

APPENDIX B. SEI CAPABILITY MATURITY MODELS

The Software Capability Maturity Model (SW-CMM®)[1] and the Capability Maturity Model Integrated (CMMI®)[2] were developed under the auspices of the Software Engineering Institute (SEI) of Carnegie Melon University.

The SW-CMM is a model for judging the maturity of the software processes of an organization and for identifying the key practices that are required to increase the maturity of these processes. This model is one of the three that provide the basis for the initial CMMI® product suite.

The SW-CMM® has become a de facto standard for assessing and improving software processes. Through the SW-CMM®, the SEI and community have put in place an effective means for modeling, defining, and measuring the maturity of the processes used by software professionals.

The Capability Maturity Model® Integration (CMMI®) project is a collaborative effort to provide models for achieving product and process improvement. The primary focus of the project is to build tools to support improvement of processes used to develop and sustain systems and products. The output of the CMMI project is a suite of products, which provides an integrated approach across the enterprise for improving processes, while reducing the redundancy, complexity and cost resulting from the use of separate and multiple capability maturity models (CMM®s).

Under both SW-CMM® and the CMMI® organizations are ranked at maturity levels 1 – 5 depending on the maturity of the organization. The higher the level, the more mature the organization. Maturity is determined by evaluating the organization to determine what proven best practices have been adopted by the organization in order to acquisition, development, upgrade and maintenance of systems and software by the organization. The CMMs do not tell an organization how to implement these practices; they simply specify what practices must be in place in order to reach a specific maturity level.

Determination of an organizations maturity level is done by an independent organization licensed by the SEI to conduct maturity level assessments by the SEI. Assessments done internally to the organization are considered to be informal and do not justify the claiming or a maturity level by the assessed organization.

1. A listing of all SEI certified lead assessors for CMMI can be found at http://www.sei.cmu.edu/managing/scampi.html.
2. A listing of all SEI certified lead assessors for SW-CMM can be found at http://www.sei.cmu.edu/managing/assessors.html.
3. Information on the most recent results of SW-CMM and CMMI assessments are available at http://www.sei.cmu.edu/sema/profile.html.

CMM LEVELS

MATURITY LEVEL	CHARACTERISTICS	RESULT
5 OPTIMIZING	• Improvement fed back into the process • Automated tools used to identify weakest process elements • Numerical evidence used to apply technology to critical tasks • Rigorous defect-casual analysis and defect prevention	P Q R U O A
4 MANAGED	**(Quantitative)** • Measured process • Minimum set of quality and productivity measurements • Process data stored, analyzed, and maintained	D L U I C T
3 Defined	**(Qualitative)** • Process defined and institutionalized • Software Engineering Process Group leads process improvement	T Y I V I
2 REPEATABLE	**(Intuitive)** • Process dependent on individuals • Basic project controls established • Strength in doing similar work, but new challenges present major risk • Orderly framework for improvement lacking	T Y R Y I
1 INITIAL	**(Ad hoc/chaotic process)** • No formal procedures, cost estimates, project plans • No management mechanism to ensure procedures are followed • Tools not well integrated; change control is lax • Senior management does not understand key issues	S K

Figure B-1: SEI CMM Levels

[1] Capability Maturity Model® for Software (SW-CMM®)
[2] Welcome to the CMMI® Web Site

APPENDIX C. MIL-HDBK-881 SECTION 2.2.5 AVOIDING PITFALLS IN CONSTRUCTING A WORK BREAKDOWN STRUCTURE

2.2.5 Avoiding Pitfalls in Constructing a Work Breakdown Structure

A sound work breakdown structure clearly describes what the program manager wants to acquire. It has a logical structure and is tailored to a particular defense materiel item. It can tie statement of work, CLIN structure, and the system description documents together. Remember: the work breakdown structure is product oriented. It addresses the products required, NOT the functions or costs associated with those products.

Elements not to include

The following paragraphs expand the explanation of what elements are to be excluded from the WBS elements:

Do not include elements, which are not products. A signal processor, for example, is clearly a product, as are mock-ups and Computer Software Configuration Items (CSCIs). On the other hand, things like design engineering, requirements analysis, test engineering, aluminum stock, and direct costs, are not products. Design engineering functional efforts; aluminum is a material resource; and direct cost is an accounting classification. Thus none of these elements are appropriate work breakdown structure elements.

Program phases (e.g. design, development, production, and types of funds, or research, development, test and evaluation) **are inappropriate as elements in a work breakdown structure.**

Rework, re-testing and refurbishing are not separate elements in a work breakdown structure. They should be treated as apart of the appropriate work breakdown structure element affected.

Non-recurring and recurring classifications are not work breakdown structure elements. The reporting requirements of the CCDR will segregate each element into its recurring and non-recurring parts.

Cost saving efforts such as total quality management initiatives, could cost, and warranty are not part of the work breakdown structure. These efforts should be included in the cost of the item they affect, not captured separately.

Do not use the structure of the program office or the contractor's organization as the basis of a work breakdown structure.

Do not treat costs for meetings, travel, computer support, etc. as separate work breakdown structure elements. They are to be included with the work breakdown structure elements with which they are associated.

Use actual system names and nomenclature. Generic terms are inappropriate in a work breakdown structure. The work breakdown structure elements should clearly indicate the character of the product to avoid semantic confusion. For example, if the Level 1 system is Fire Control, then the Level 2 item (prime mission product) is Fire Control Radar.

Treat tooling as a functional cost, not a work breakdown structure element. Tooling (e.g., special test equipment, and factory support equipment like assembly tools, dies, jigs, fixtures, master forms, and handling equipment) should be included in the cost of the equipment being produced. If the tooling cannot be assigned to an identified subsystem or component, it should be included in the cost of integration, assembly, test, and checkout.

Include software costs in the cost of the equipment. For example, when a software development facility is created to support the development of software, the effort associated with this element is considered part of the CSCI it supports or, if more than one CSCI is involved, the software effort should be included under the integration, assembly, test, and checkout. Software developed to reside on specific equipment must be identified as a subset of that equipment.

APPENDIX D. MIL-HDBK-881 SECTION 3.2 CONTRACTUAL ISSUES AND SECTION 3.2.1 SOFTWARE AND SOFTWARE INTENSIVE SYSTEMS

3.2 CONTRACTUAL ISSUES

The contractor's expanded work breakdown structure must address all Program WBS elements. Contractors should include lower breakdown levels where they identify risks associated with technical issues or resources, and identify control plans whether or not the items are reported back to the government. For example, software development tends to be high technical risk and high cost. Since all software that is an integral part of any specific equipment system and subsystem specification or specifically designed and developed for system test and evaluation should be identified with that system, subsystem, or effort, it may be appropriate to collect lower level information when it exists. In such cases, the following structure and definitions could be used:

LEVEL 4	LEVEL 5
Build 1…n (Specify names)	CSCI 1…n (Specify names) CSCI to CSCI Integration and Checkout

Integration, Assembly, Test and Checkout

3.2.1 Software and Software Intensive Systems

The importance of software in today's government acquisition environment is growing. As a result software is identified in two ways for development of a work breakdown structure: the first type of software is that which operates or runs on a specific piece of equipment, and the second type of software is that which may be contracted for separately from the operating equipment or is a stand alone (software intensive system). Software that is being developed to reside on specific equipment must be identified as a subset of that equipment. Multi-function software will be identified as a subset of the equipment work breakdown structure element, which either includes the software in the element specification or exercises the most critical performance constraint. Refer to Figure 3-1 for an example of how software should be addressed as part of specific equipment. In cases where the application of this rule results in a conflict in the selection of the proper element, the specification relationship will take precedence. For example, an aircraft's electronic equipment typically has software included in each of the subsystem elements. Software that resides and interfaces with more than one equipment, i.e., applications software, and overall system software which facilitates the operation and maintenance of the computer systems and associated programs (e.g., operating systems, compilers, and utilities) will be called out at the appropriate work breakdown level within the program.

It is incorrect to summarize all software on a program or contract in a work breakdown structure. By separating these elements from the hardware they support, performance measurement and management control over each equipment is difficult to maintain. The true cost of each

equipment is not readily available for decision concerning that equipment. Rather than separately summarizing software, it is important to identify software with the hardware it supports. (When needed, a contractor's management system can use an identifier for each software element to produce summaries for software management purposes.)

A separately contracted or stand alone software will include the software, data, services, and facilities required to develop and produce a software product for a command and control system, radar system, information system, etc. Where software is considered stand-alone (i.e., does not reside or support a specific equipment, or is considered a pure software upgrade, etc.), the government should use the same product-oriented work breakdown structure format. Figure D-1 provides an example of a work breakdown structure for a stand alone software system.

SOFTWARE INTENSIVE SYSTEM WBS
1 SOFTWARE INTENSIVE SYSTEM
2 PRIME MISSION PRODUCT
3 APPLICATIONS SW
4 BUILD 1
5 CSCI 1…n
5 CSCI TO CSCI INTEG. AND CHKOUT
4 BUILD 2…n
5 CSCI 1…n
5 CSCI TO CSCI INTEG. AND CHKOUT
4 APPLICATIONS S/W INTEG, ASSEMBLY, TEST, & CHKOUT
3 SYSTEM SW
4 BUILD 1
5 CSCI 1…n
5 CSCI TO CSCI INTEG. AND CHKOUT
4 BUILD 2…n
5 CSCI 1…n
5 CSCI TO CSCI INTEG. AND CHKOUT
4 SYSTEM S/W INTEG, ASSEMBLY, TEST, & CHKOUT
3 INTEG. ASSEMBLY, TEST AND CHECKOUT
3 HW/SW INTEGRATION
2 SYSTEMS ENGINEERING/PROGRAM MANAGEMENT
2 SYSTEMS TEST AND EVALUATION
2 TRAINING
2 DATA
2 PECULIAR SUPPORT EQUIPMENT
2 COMMON SUPPORT EQUIPMENT
2 INITIAL SPARES AND REPAIR PARTS

Figure D-1: Example of Software Intensive System WBS

APPENDIX E. SOFTWARE IN THE WORK BREAKDOWN STRUCTURE

Contracts With Hardware and Software

Software that is being developed to reside on specific equipment must be defined as a subset of that equipment. Multi-function software will be identified as a subset of the equipment work breakdown structure element, which either includes the software in the element specification or exercises the most critical performance constraint. In cases where the application of this rule results in a conflict in the selection of the proper element, the specification relationship will take precedence. For example, an aircraft's electronic equipment typically has software included in each of the subsystem elements. Software that resides and interfaces with more than one equipment, i.e., applications software, and overall system software which facilitates the operation and maintenance of the computer systems and associated programs (e.g., operating systems, compilers, and utilities) will be called out at the appropriate work breakdown level with the program (ref. ANSI/IEEE Std 610.12 for definitions of applications and system software).

It is incorrect to summarize all software on a program or contract in a work breakdown structure. By separating these elements from the hardware they support, performance measurement and management control over each equipment is difficult to maintain since the true cost of each equipment is not readily available. Rather than a separate summarization, software should be identified with the hardware it supports. (When needed, contractor management systems can use an identifier for each software element to produce internal summaries for software management purposes).

Software Only Contracts

Separately contracted or stand alone software will include the software, data, services, facilities required to develop and produce a software product for a command and control system, radar system, information system, etc.

APPENDIX F. SAMPLE SOFTWARE WORK BREAKDOWN STRUCTURE

This sample WBS was taken from "A Practical Guide To Estimating Software Cost", Version 1.2, prepared jointly by the Software Engineering Division and NAVAIR Cost Department, 25 July 2000. Only the WBS areas for software metrics, software development, and software integration and testing are listed in their entirety. Refer to the Guide for the element breakout of other WBS areas of interest.

The following Work Breakdown Structure (WBS) is oriented to the development of a MIL-STD-498 or DoD-STD-2167A software product. The Work Breakdown Structure (WBS) is a slightly modified version of the WBS presented in the Software Estimation Process Version 2.2, [Ref 1] from the Software Engineering Process Office at the Naval Command, Control, and Ocean Surveillance Center (NCCOSC), Research, Development, Test and Evaluation Center (RDTEDIV) which is now part of the SPAWAR organization in San Diego. The process has been modified to reflect the Software Size Estimation Process presented in "A Practical Guide To Estimating Software Cost", Version 1.2, prepared jointly by the Software Engineering Division and NAVAIR Cost Department, 25 July 2000. It is highly detailed and based on the waterfall model for software development. It is meant to be tailored to each project's specific tracking needs and requirements. The WBS contains adequate detail to enable tracking tasks with duration as small as two to three weeks. This WBS is more representative of day to day management utilized by software developers and is not indicative of the level of detail that would be provided in EVM reporting. This level of detail would be representative of task management necessary to develop the Integrated Master Schedule .

PROJECT MANAGEMENT
SOFTWARE ESTIMATION
RISK MANAGEMENT
SOFTWARE METRICS
- Software Metrics plan
  - Tailor basic metrics process
  - Define risk metrics process
- Update/revise Metrics Plan
- Track/analyze cost/schedule variance
- Track/analyze progress
- Track/analyze code size
- Track/analyze documentation size
- Track/analyze requirements testability
- Track/analyze requirements traceability
- Track/analyze Engineering Change Proposal (ECP)/System Change Notices (ECPs)
- Track/analyze build/release content
- Track/analyze staffing
- Track/analyze computer resource utilities
- Track/analyze defects
- Formal reports

Report final project analysis
SYSTEM ENGINEERING
CONFIGURATION MANAGEMENT
SOFTWARE QUALITY ASSURANCE
SOFTWARE DEVELOPMENT PLAN (SDP)
INTERFACE REQUIREMENTS
DATABASE REQUIREMENTS
DEVELOPMENTAL SOFTWARE
Analyze system requirements
Identify software requirements
Determine derived software requirements
Identify candidate COTS software
Identify candidate reusable software
Perform feasibility studies * 1 days
Select computer language(s)
Allocate functions/identify CSCIs
Determine software requirement testability
CSCI 1 - N
FUNCTIONAL REQUIREMENTS
Analyze CSCI requirements
Preliminary Software Requirements Specification(SRS)
Identify internal interfaces
Identify functional/derived requirements
engineering
data elements
safety
security
human engineering
Identify software quality factors
Identify design constraints
Identify qualification methods
Trace requirements to SSS
SRS inspection(s)/review(s)
Software Specification Review
Update SRS
Baseline SRS
PRELIMINARY DESIGN
Preliminary Design analysis
Identify Software Units (SU)
Identify internal interfaces
Identify external interfaces
Preliminary System Design Document (SDD)
Overview
Architecture
Memory/processing time allocation
CSCI design description

- SU 1 - N
  - Identify allocated requirements
  - Identify SUs
  - Identify relationships between SUs
  - Data flow and execution control
  - Identify derived requirements
  - Trace requirements to SRS
  - SU inspection(s)/walkthrough(s)
  - SU design rework
- Preliminary SDD inspection(s)/review(s)
- Preliminary Design Review
- Update Preliminary SDD

DETAILED DESIGN

- Detailed SDD
  - SU 1 - N
    - Describe constraints
    - Describe input/output data elements
    - Describe local data elements
    - Describe interrupts and signals
    - Describe algorithms
    - Describe data structures
    - Describe local datafiles/database
    - Describe limitations
    - Trace requirements to Preliminary SDD
    - SU inspection(s)/walkthrough(s)
    - SU design rework
- SDD inspection(s)/review(s)
- SDD rework
- Critical Design Review
- Update SDD
- Baseline SDD

CODE and UNIT TEST

- SU 1 - N
  - Design/document unit test
  - Code and compile
  - Write comments/header
  - Code inspection(s)/walkthrough(s)
  - Rework
  - Testing and analysis
  - Rework
  - Maintain SDF
  - Turn over accepted SU to CM

TEST READINESS REVIEW

SU INTEGRATION and TESTING

- Analyze Software Test Report
- Perform necessary rework

- Perform SU regression testing
- Update SDFs
- TEST READINESS REVIEW
- CSCI INTEGRATION and TESTING
  - Analyze Software Test Report
  - Perform necessary rework
  - Perform SU regression testing
  - Update SDFs

SOFTWARE INTEGRATION & TESTING

- Software Test Plan
  - Determine general test requirements
  - Determine test classes
    - stress
    - timing
    - erroneous input
    - maximum capacity
  - Determine test levels
    - CSCI
    - CSCI to CSCI integration
    - CSCI to Hardware CI (HWCI) integration
    - system
  - Determine test definitions
      - determine objective
      - determine special requirements
      - identify test type/class
      - determine qualification method
      - cross reference to SRS requirements
      - determine type of data to record
      - identify assumptions/constraints
      - determine test schedule
      - identify data analysis techniques
- Perform Integration & Testing
  - System
    - Integrate CSCIs
    - Write System Test Description
    - Conduct Test Readiness Review
    - Perform testing and analysis
    - Write System Test Report
    - Rework
    - Regression testing
    - CSCI
      - Integrate SUs
      - Write Software Test Description
      - Conduct Test Readiness Review
      - Perform testing and analysis
      - Write Software Test Report

Rework
Regression testing
SU
    Integrate SUs
    Write Software Test Description
    Conduct Test Readiness Review
    Perform testing and analysis
    Write Software Test Report
    Testing and analysis
    Rework
    Regression testing

APPENDIX G. COCOMO II ESLOC

As an example of a formula for calculating ESLOC, we will discuss the COCOMO II ESLOC formula, which is provided in the public domain[1,2]. As stated previously, there are many different versions and types of ESLOC formula; the COCOMO II formula provides a good example of the various considerations for such a formula.

$$\text{Equivalent KSLOC} = \text{Adapted KSLOC} \times (1 - {}^{\text{AT}}\!/_{100}) \times AAM$$

$$AAF = (0.4 \times DM) + (0.3 \times CM) + (0.3 \times IM)$$

$$\textbf{AAM} = \begin{cases} \left[\dfrac{AA + AAF \times (1 + (0.02 \times SU \times UNFM))}{100} \right], for\ AAF \leq 50 \\ \left[\dfrac{AA + AAF + (SU \times UNFM)}{100} \right], for\ AAF > 50 \end{cases}$$

Percentage values when entered in the formula should be as a percentage not as fractional or decimal value. If AT is 23% use 23 not .23 in the formula.

Definitions for terms in the previous formulas follow below.

Adapted KSLOC – Number of KSLOC of code to be modified, reused or automatically generated.

AAM – Adaptation Adjustment Modifier.

AAF – The amount of modification. If DM, CM and IM are 100% (new code) then AAF will be 100%.

AT – Automatically Translated, the percentage of modified software, which will be automatically translated or converted for use in the system. The COCOMO II formula does not attempt to include automatically translated code as part of its ESLOC. It is used only to subtract the automatic translation effort from the remainder of the development. It must be estimated separately.

DM – Is the percentage of the adapted software's design that is modified in order to adapt it to the new requirements of the environment.

CM – Is the percentage of the adapted software's code that is modified in order to adapt it to the new requirements of the environment. DM will probably be greater than or equal to CM. If it's not, it raises questions into the quality of the modification effort. If adequate upfront redesign is not performed, than the code modification becomes little more than a hacking effort with subsequent low quality and much more extensive testing requirements.

IM – Is the percentage of effort required to integrate the adapted software into an overall product and to test the resulting product as compared to the normal amount of integration and test effort for software of comparable size. Expect IM to be greater than DM and CM. Not only must the modified design and code be tested, but in addition the integration of that modified code to the unmodified code must also be verified. For complex applications that have been extensively modified, IM can be in access of 100%. Even if CM and DM are 0 (completely unmodified code), IM will still be in excess of 0 since the integration of the unmodified reused code with the remainder of the system must still be verified. In this case IM should be relatively low.

SU – Software Understanding Increment. Expressed as a percentage obtained from Table G-1 below. If the software is rated very high on structure, applications clarity and self-descriptiveness, SU is 10%. If the software is rated very low on these factors SU is 50%. SU is determined by taking the subjective average of the three categories. The SU identifies how hard it will be for the developers attempting to modify the code to understand the code. The harder it is to understand the higher the SU, which means the harder it will be to modify which will in turn increase the ESLOC. If the code is being reused with no modification (DM = 0, CM = 0), SU is set to 0.

	Very Low	Low	Nominal	High	Very High
Structure	Very low cohesion, high coupling, spaghetti code	Moderately low cohesion, high coupling	Reasonably well-structured; some week areas	High cohesion, low coupling	Strong modularity, information hiding in data/control structures
Application Clarity	No match between program and application world-views	Some correlation between program and application	Moderate correlation between program and application	Good correlation between program and application	Clear match between program and application
Self Descriptiveness	Obscure cold; documentation missing, obscure or obsolete	Some code commentary and headers; some useful documentation	Moderate level of code commentary	Good code commentary and headers; useful documentation; some week areas	Self-descriptive code; documentation up-to-date, well organized, with design rational
SU Increment to ESLOC	50	40	30	20	10

Table G-1: Rating Scale for Software Understanding Increment (SU)

AA – Assessment and Assimilation, determines whether a reused software module is appropriate to the application, and to integrate its description into the overall product description. Table G-2.

AA Increment	Level of AA Effort
0	None
2	Basic Module Search and documentation
4	Some module Test and Evaluation (T&E) documentation
6	Considerable module T&E, documentation
8	Extensive module T&E documentation

Table G-2: Rating Scale for Assessment and Assimilation Increment (AA)

UNFM – Programmers relative unfamiliarity with the software. See Table G-3 for guidance on determination of UNFM. If the programmer works with the software every day, the 0.0 multiplier for UNFM will add no software understanding increment. If the programmer has never seen the software before, the 1.0 multiplier will add the full software understanding effort increment. If the software is being used unmodified (DM = 0, CM = 0) then UNFM = 0.

UNFM Increment	Level of Unfamiliarity
0.0	Completely familiar
0.2	Mostly familiar
0.4	Somewhat familiar
0.6	Considerably familiar
0.8	Mostly unfamiliar
1.0	Completely unfamiliar

Table G-3: Rating Scale for Programmer Unfamiliarity (UNFM)

Figure G-1 shows how the AAM, (Adaptation Adjustment Modifier) changes with respect to AAF, the percentage of modification. The figure reflects how changes in: the quality of the code, documentation quality, developer understanding, and similarity of the domain of the code to be modified to the new system requirements impact the ESLOC. Since AAM is multiplied by the amount of adapted code, it shows how these factors affect the equivalent size of the code. In the figure a best case and a worse case scenario are plotted. Notice the equivalent SLOC for the worse case, where developer understanding is low, code quality is low, documentation is poor and the domain similarity low is much higher than it is for the best case scenario. The worse case scenario shows a break-even point of about 46%, above which the amount of ESLOC will

actually be greater than the amount of adapted code. This is the break-even point at which it becomes easier to develop the code from scratch rather than perform modification.

One flaw in the COCOMO II ESLOC equations is, if the developer is very familiar with the code, UNFM is zero. Then it doesn't make any difference how poorly written and documented the code is. In this case the COCOMO II ESLOC formula will arrive at the same ESLOC for both poorly written and documented code as it would for well-written and documented code. This is false. No matter how familiar the developer is with the code, low quality code is inherently more difficult to modify and test than high quality code. In this regard, the COCOMO II ESLOC formula underestimates the difficulty of modifying such low quality code.

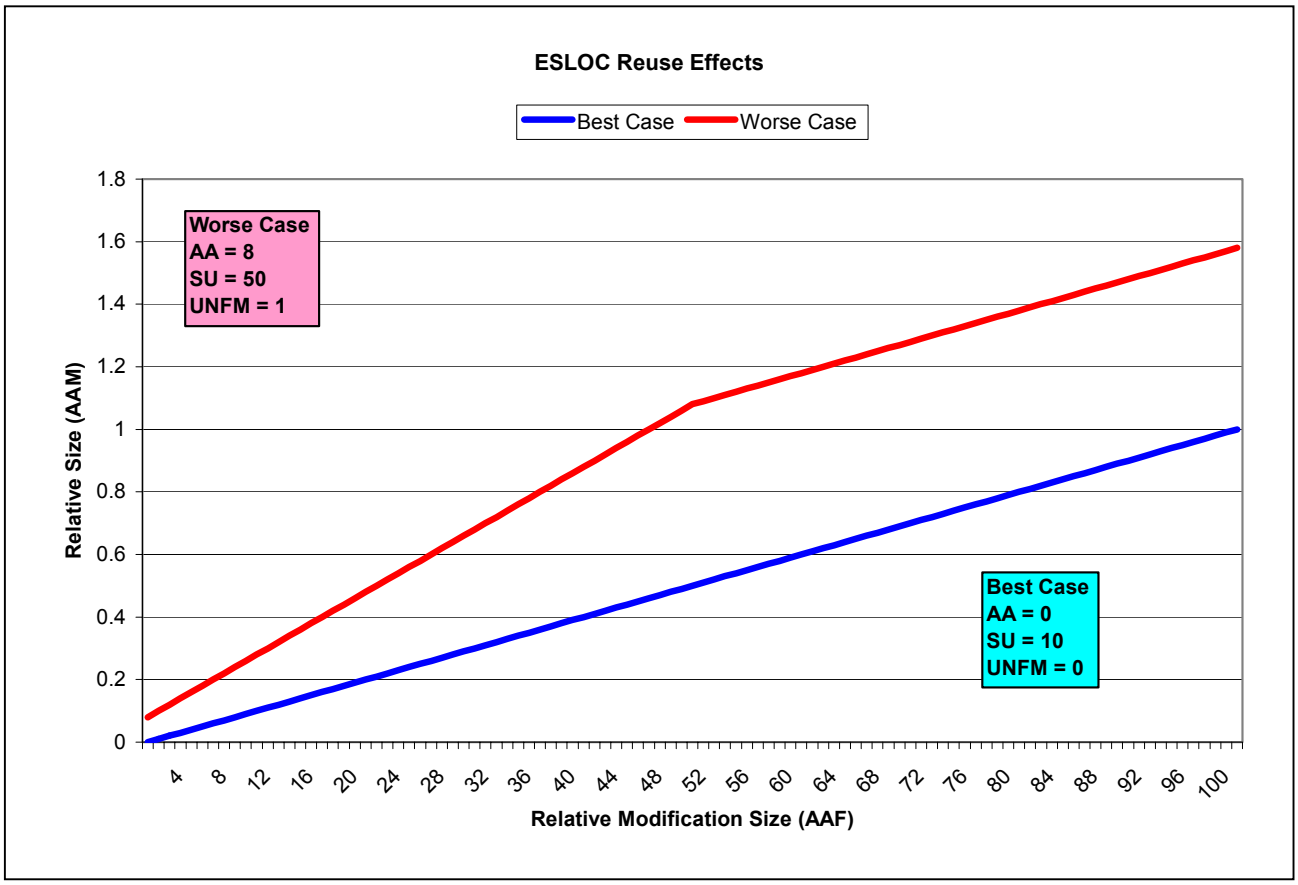


Figure G-1: ESLOC Reuse Effects

[1] Software Estimation With COCOMO II, Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald Reifer, Bert Steece, Prentice Hall PTR, 2000, ISBN 0-13-026692, page 19 - 27

[2] COCOMO II Model Definition Manual, page 21 - 26, http://sunset.usc.edu/research/COCOMOII/index.html

APPENDIX H. Technical Performance Measurements (TPM)

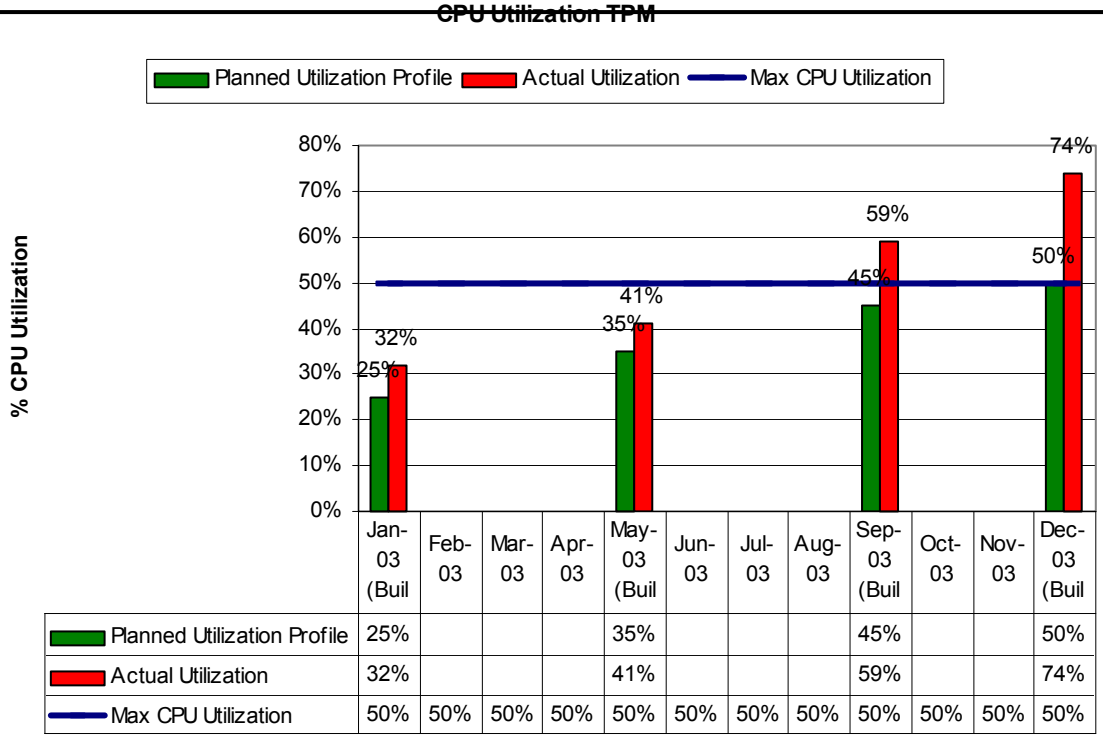
TPMs are used to measure progress in achieving the technical objectives of the system. TPMs are phased over time in order to judge the progress in meeting Key Performance Parameters (KPP), which are usually associated with measurable performance or capacity requirements.

As applied to software, a TPM could estimate what amount of capacity or performance will be achieved in early releases if the specified capacity or performance requirement is to be met in the final release. For these earlier builds or releases, the estimated capacity or performance will be lower than the final requirement, since these early releases include only a subset of the total final functionality. In later releases, the TPM increases permissible capacity or performance allowed until in the final release it reaches the level of the KPP identified in the appropriate requirements document. If at some point a software release is unable to meet its TPM requirement, the TPM provides an early warning that the system has a problem that may prevent it from reaching the KPP. If a TPM is not used, these potential capacity and performance requirements problems might not be detected until late in the development cycle when it would be extremely difficult to correct them in time to meet the project schedule and correcting them would be very costly.

The system or software requirements may identify a desired performance level in addition to a minimally acceptable performance level. In this case a TPM may also include upper and lower acceptable bounds for the requirement in question. For earned value purposes the lower bound or minimally acceptable performance should be used as the basis for allocating BCWP. If a higher performance level than the minimum acceptable is used, the project's CPI and SPI will show unsatisfactory progress even if the minimum acceptable performance is being achieved. Desired performance in excess of the minimum acceptable should be rewarded via performance awards rather than used in earned value calculations.

Figure H-1 is an example of a software TPM. In this case there is a capacity requirement for the software to use no more than 50% of the total processing capacity of CPU "A" once the software is completed. Further assume that there are four software releases or builds planned for the development of the software. Each consecutive build adds more of the systems requirements and utilizes a higher percentage of the CPUs processing capacity. The TPM for CPU "A's" processing requirement might then specify that no more than 25% of CPU "A's" processing capacity be used at the completion of build 1, 35% at the completion of build 2, 45% at the completion of Build 3, and 50% at the end of build 4. Thus if the software is using 25% of CPU "A's" processing capacity at the completion of Build 1, it would be meeting its TPM. If over 25% of the processing capacity is being used, this would indicate that some corrective action needs to be taken in order to ensure the final 50% CPU "A" processing requirement is to be met. This TPM is "one sided", if the CPU utilization is greater than specified in the TPM, this indicates a problem which the earned value should reflect, if it's lower than the TPM value for the point in time, this is good, but will not cause any improvement in the earned value beyond meeting the TPM.

Figure H-1: Example CPU Utilization TPM



	Jan-03 (Buil	Feb-03	Mar-03	Apr-03	May-03 (Buil	Jun-03	Jul-03	Aug-03	Sep-03 (Buil	Oct-03	Nov-03	Dec-03 (Buil
Planned Utilization Profile	25%				35%				45%			50%
Actual Utilization	32%				41%				59%			74%
Max CPU Utilization	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%

Failure to meet TPMs must be taken into account when determining earned value since they indicate that the planned for progress in meeting the programs performance, and therefore its cost and schedule goals are not being achieved. This being the case, an effective EVM system must identify these problems so that corrective action can be taken.

Earned Value Determination

The software implementing a requirement cannot be considered complete and full-earned value awarded until any capacity or performance requirements affected by the implementation of that requirement are also met. For example: there is a requirement to allow the operator to enter a latitude and longitude that will than be marked on the systems map display. Assume the software correctly implements this functionality. However, the RAM utilized by the software, which includes this requirement, is 60% utilized when according to the TPM for the RAM it should be only 40% utilized. This latitude/longitude requirement's implementation is not complete. Additional effort and time must be expended to revise the system or software so that the RAM's TPM is met. There will also be many other requirements, which are also affected by this TPM and cannot be considered to have been completely and correctly implemented. Therefore all of the earned value for completing the requirements that must execute in the RAM covered by this TPM cannot be earned until the TPM is met. Some considerations for using TPMs as a basis for earned value are:

1. The project needs a well-defined program Work Breakdown Structure (WBS) that is directly associated with the KPPs of the system being designed, and has clear links to the associated EVMs control accounts[1]. For example: if there is a utilization requirement on how much of the processing capacity of a CPU can be utilized, the WBS should be structured so that this KPP can be directly associated with the software executing on that CPU.

2. Time phased TPMs must be established prior to the Integrated Baseline Review (IBR)[10]. It may be difficult to develop these TPMs at this phase of the development, but it is necessary if they are to be used to determine earned value and ensure software related KPPs are being met. As the systems and software design develops, TPMs may be adjusted and corrected based on more complete information.

Note - While this may be a difficult task, not establishing TPMs essentially means that the program has decided to ignore the risk of not meeting the KPP associated with the TPM and makes no effort to track its progress. This means if the KPP is not achieved, the problem will not be detected until late in the software integration or systems testing when it will be extremely difficult to correct and there will be little schedule and funding with which to correct it. Earned value cannot warn of a problem, unless a measure that calibrates earned value to the problem is used as its basis.

3. The progress of software in meeting its TPMs is most effectively measured during systems testing, or any test phase where the software that the TPM is associated with, is tested on its actual target software. This means that TPMs are most likely to be useful in a software development when an incremental, evolutionary, or spiral development lifecycle is being implemented. These lifecycle models provide multiple test phases in which time phased releases of the software, each implementing progressively more of the planned for functionality can be measured. A Waterfall lifecycle model on the other hand essentially provides only a single opportunity to evaluate the software against the associated TPM.

4. By the time it is discovered that a software release has not achieved its TPMs performance requirement, BCWP for that release's requirements analysis, design, and code & unit test phases will have already been earned and allocated based on the implementation of functional requirements. Rather than attempt to correct the BCWP for these previous phases, instead the BCWP should be reduced for the test phase in which the problem was discovered and in all subsequent phases until the next opportunity to evaluate the software's progress in meeting the TPM.

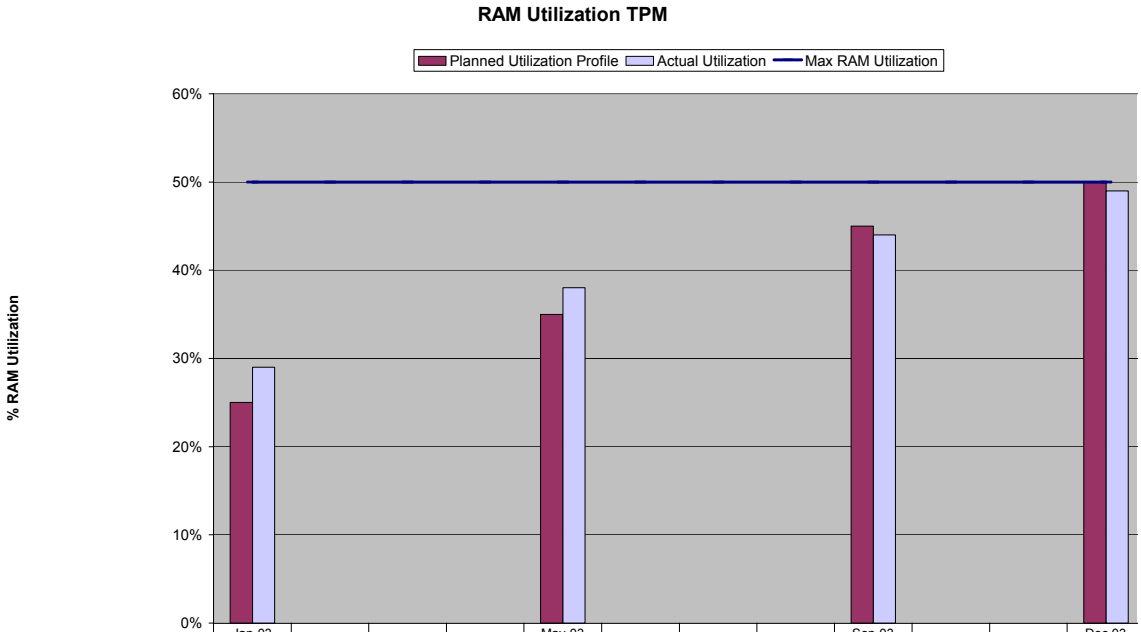
If TPMs have been established for a software development, how can they be integrated with earned value in order to insure deviations from the TPM are reflected? Ferraro 2002[10], Kulick 97 and 98[2], and Coleman, Kulick and Pisano 1996[3] discuss different means of calibrating TPMs with earned value. Each of these methods utilizes similar approaches of varying degrees of complexity. None of these methods are specifically developed for use with software but as generic methods of applying TPMs to earned value in systems. The following discussion and example will utilize the method discussed by Ferraro 2002[10]. This is the least complex implementation. Considering the accuracy of the various assumptions and estimates likely to be made in developing the TPM and its impact on EVM, Ferraro's method provides a similar level of accuracy to more complex methods. Further, the simpler method discussed by Ferraro 2002[10], which does not consider Technical Readiness Levels (TRL) is used since TRLs are not applicable to software development. The following TPM earned value discussion will only consider its application to software performance and utilization KPPs.

Following are the steps to adjust earned value based on TPMs:

1. Determine BCWP, ACWP, CPI and SPI for the WBS without considering TPM impact. In other words, calculate the CPI and SPI using the same methods as would be utilized if no TPMs existed.
2. Determine each TPM's impact on the WBS element. TPM impact is a percentage, which when summed with the coverage of all TPMs for the WBS element and the percentage not covered by the TPMs comes to 100%. The percentage not covered by the TPMs will be referred to as "Other". The Other category can be considered to be that part of the development directly related to developing the functional requirements rather than the utilization and performance requirements. This value will be determined by analysis on the part of project systems and software engineers.
3. Determine the BCWP affected by the TPM by multiplying the TPMs impact by the BCWP. The sum of the BCWP affected for each TPM and the Other category will be the BCWP for the WBS element.
4. Determine the TPM Technical Score. This is 100% minus the percentage of deviation from the appropriate value on the time phased TPM. The technical score for the Other category is always 100%. Maximum value for the technical score is 100%. So if the software's performance exceeds the TPM requirement, its technical score is no better than if it had met it. Rather than rewarding the developer with more earned value for exceeding performance requirements, which may obstruct the ability of EVM to highlight other problems, consider using award fees to encourage the developer to exceed KPPs.
5. Multiply the TPM Technical Score by the BCWP affected to arrive at the New BCWP or BCWP that is "TPM informed".
6. Sum the New BCWP for each TPM and the Other category to arrive at the New BCWP for the entire WBS.
7. Calculate New CPI and SPI for the WBS based on the New BCWP for the WBS.
8. Determine Composite Technical Score by dividing New BCWP by BCWP.

Example:

In this example we will apply two TPMs, one for CPU utilization, Figure H-1 and one for RAM utilization, Figure H-2 to adjust the earned value for the CSCI A that utilizes these processing resources.



	Jan-03 (Build 1)	Feb-03	Mar-03	Apr-03	May-03 (Build 2)	Jun-03	Jul-03	Aug-03	Sep-03 (Build 3)	Oct-03	Nov-03	Dec 03 (Build 4)
Planned Utilization Profile	25%				35%				45%			50%
Actual Utilization	29%				38%				44%			49%
Max RAM Utilization	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%

Figure H-2: RAM Utilization TPM

WBS			January 03 TPMs	TPM Impact (effect on and coverage of WBS)	BCWP Affected by TPMs	TPM T.S.	New BCWP "TPM Informed"
	Contract						
CSCI A Build 1 System Test	**Current CUM ($K)**	**New ($K)**	CPU Utilization	25%	$25	72.00%	$18
			RAM Utilization	25%	$25	84.00%	$21
BAC	$100		TOTAL	50%	$50		$39
BCWP	$100	$89					
BCWS	$100		Other (Not Affected	50%	$50	100.00%	$50
ACWP	$120		by TPMs)				
CPI	83%	**74%**	New TOTAL	100%	$100		$89
SPI	100%	**89%**		Composite Technical Score		89%	

Table H-1: EVM Adjustment Following CSCI A Build 1 System Test

In the Table H-1 "Current CUM" column we have earned value unadjusted for TPM performance. In our example the developer has achieved all of his CSCI A Build 1 System Test objectives as far as verifying functional requirements implementation. BCWP and BCWS are equal. However these numbers do not reflect the impact of the developer not being entirely successful in achieving the TPM objectives for RAM and CPU utilization requirements. Even though testing showed that the functional requirements were correctly implemented, their implementation is using more CPU processing capacity and RAM than planned for. Thus the BCWP must be reduced to account for these requirements not being correctly implemented and the project not having achieved its planned objectives. This is done using the previous eight

steps, with the adjusted BCWP appearing in the "New" column. In our example the project systems and software engineers have determined that the CPU Utilization and RAM Utilization TPMs each have a 25% impact on the System Test and subsequent phases. The further the actual values of CPU and RAM utilization deflect from the planned values in their TPM's the lower the TPM Technical Score and the New BCWP, which results in a lower New CPI and SPI.

Notice how in Table H-1 the New BCWP, CPI and SPI are significantly lower after being adjusted for the effect of the TPMs. What would the effect be on the technical score if the performance was better than expected? The Technical Score would be set to 100% for the TPM, since this is its maximum value. The adjustment in Table H-1 is occurring only for the CSCI Build 1 System Test. In order for the actual impact on the program of not meeting the TPM performance requirement for January 03 to be determined, the earned value must continue to be adjusted based on these results until such time as another test can be executed to determine if the corrective action has been effective. This is assuming that the current performance deficiency will continue at the same level, until empirical test data showing it has been corrected, is improving, or is worsening is available. In our example, we will assume that such a test opportunity will not occur until Systems Testing for CSCI Build 2 in May 03.

February 03		
WBS		
Contract		
CSCI A Build 2		
Req Analysis	Current CUM ($K)	New ($K)
BAC	$25	
BCWP	$25	$22
BCWS	$25	
ACWP	$25	
CPI	100%	89%
SPI	100%	89%

March 03		
WBS		
Contract		
CSCI A Build 2		
Design	Current CUM ($K)	New ($K)
BAC	$100	
BCWP	$100	$89
BCWS	$100	
ACWP	$150	
CPI	67%	59%
SPI	100%	89%

April 03		
WBS		
Contract		
CSCI A Build 2		
C&UT	Current CUM ($K)	New ($K)
BAC	$150	
BCWP	$150	$134
BCWS	$150	
ACWP	$200	
CPI	75%	67%
SPI	100%	89%

Table H-2: EVM Adjustment for CSCI A Build 2 Pre Test Phases

Since in our example, there will be no opportunity to recompute the Composite Technical Score based on new test results until CSCI A Build 2 Systems Testing, for CSCI A Build 2 Requirements, Design and Code & Unit Test phases, we will take a shortcut and compute their New BCWP for each phase by taking the Composite Technical Score arrived at in Table H-1 and multiplying it by the BCWP for the phase to arrive at the New BCWP, CPI and SPI, see Table H-2. Again this results in significant declines in the CPI and SPI in comparison with where they would be without considering the TPM. Notice that in our example ACWP exceeds BCWP in the design and C&UT phases. This is very likely to occur in such a situation as additional resources are expended in trying to correct the TPM problems noted in the previous Build 1 System Test.

WBS	Contract			May 03 TPMs	TPM Impact (effect on and coverage of WBS)	BCWP Affected by TPMs	TPM T.S.	New BCWP "TPM Informed"
CSCI A Build 2 System Test	Current CUM ($K)	New ($K)		CPU Utilization	25%	$25	82.86%	$21
				RAM Utilization	25%	$25	91.43%	$23
BAC	$100			TOTAL	50%	$50		$44
BCWP	$100	$94						
BCWS	$100			Other (Not Affected by TPMs)	50%	$50	100.00%	$50
ACWP	$150							
CPI	67%	62%		New TOTAL	100%	$100		$94
SPI	100%	94%			Composite Technical Score		94%	

Table H-3: EVM Adjustment for CSCI A Build 2 System Test

As a result of CSCI A Build 2 Systems Testing, we note an improvement in the overall Composite Technical Score, which indicates an improvement in meeting the TPMs and a subsequent reduction on the TPMs impact on EVM, see Table H-3. While the TPMs negative impact on EVM is reducing, the higher than planned ACWP, likely resulting from an effort to correct the performance deficit indicated by the TPM, is still causing the CPI and SPI to be low. In our example, notice that the New SPIs indicate the program is behind, even though it seems to be meeting its development and test schedule. This is an example of passing a schedule milestone without meeting all of the milestone requirements. In this case, the system is not meeting the utilization requirements for RAM and the CPU. At some point in the development in addition to the rising costs to correct the problem as indicated by the New CPIs, there are also likely to be schedule slips. Note that in Figure H-1, the CPU utilization is 74% in build 4 while the maximum value is 50%. If this problem is to be corrected, the schedule will have to slip.

What priority defect should this problem be considered? The closer CPU and RAM utilization gets to 100%, the more likely it will impact other performance requirements. The system may no longer be able to respond quickly to operator inputs. It may start to loose essential navigation and contact data. In this situation, it is a priority 1 or 2 defect. If the utilization is still well below 100%, even though it still exceeds system requirements, it may be considered a priority 3 defect. It will have an impact on future costs of maintenance and the ability to add new capabilities to the system, but it is not preventing the system from meeting its current performance requirements. In this case, program management may decide to relax the requirement and accept higher maintenance costs in the future and hope technology advances allow for a correction of the problem at a later date.

In the previous example, relaxing the performance or utilization requirement looks like the ideal way of solving the problem since it immediately negates the impact of not meeting the TPM at no cost. ***THIS SHOULD BE DONE WITH EXTREME CAUTION!*** If the requirement is relaxed early in the program, then there is no incentive for the developer to attempt to correct the problem nor will the earned value indicate there is a problem. However, if the problem continues to worsen it may reach a point where relaxing the requirement is no longer an option due to the impact on the implementation of essential mission requirements, making it a priority 1 or 2 defect. In this case the time earlier in the development when the developer could have been working to mitigate or

eliminate the problem has been lost along with any emphasis to control this issue. Thus relaxing the requirement early in the effort may actually aggravate the problem. Instead, relaxing the requirement should not be done until late in the development thus continuing to emphasize to the developer the importance of working to reduce this problem throughout the development cycle. This increases the odds that the difference between the planned and actuals for these performance and utilization requirements will be small enough so they will not impact essential mission requirements and preserve the option of reducing the performance or utilization requirement prior to OPEVAL.

NOTE – RELAXATION OF PERFORMANCE AND UTILIZATION REQUIREMENTS SHOULD BE DONE AS LATE AS POSSIBLE IN THE PROGRAM IN ORDER TO PRESERVE THE EMPHASIS ON MEETING THESE REQUIREMENTS AND INSURING THAT ANY PROBLEMS DO NOT REACH UNACCEPTABLE LEVELS THROUGH NEGLECT.

The methods of quantitatively correlating TPMs with EVM discussed above have only been tried in a limited number of cases and have had limited success in predicting final results. These problems were at least partially caused by the pilot programs having poorly structured WBSs, poorly defined TPMs and loss of visibility due to rebaselinings of the program. While the method of quantitatively correlating EVM and TPMs may not be perfect, the alternative is to not consider KPPs in project EVM calculations. ***THIS MEANS EVM WILL GIVE NO WARNING OF SUCH PROBLEMS, AND WILL BE UNABLE TO PREDICT THE RESULTS OF THE PROGRAMS NOT MEETING THESE KEY PERFORMANCE PARAMETERS.*** Even a methodology with a few flaws is better than this alternative. Refusing to use TPMs in EVM because of these possible flaws is similar to refusing to wear a parachute when skydiving because it doesn't 100% guarantee your safety.

As with any measurement method, if an adverse prediction is acted upon and corrected early, you won't see the effect, which is of course what is desired. Some may then argue that the 'prediction' was not accurate and the correction therefore unnecessary. This is the same as arguing that steering corrections are unnecessary because the car didn't go off the road.

[1] "Technical Performance Measures – A Program Managers Barometer", Mike Ferraro, Program Manager Magazine, November – December 2002, http://www.dau.mil/pubs/pm/pmpdf02/Nov_Dec/NO-DE02.pdf

[2] "Technical Performance Measurement – A Systematic Approach to Planning, Integration and Assessment – Part I", "Technical Performance Measurement – A Systematic Approach to Planning, Integration and Assessment – Part II – Linking Technical Performance to Cost and Schedule", and "Technical Performance Measurement – A Systematic Approach to Planning, Integration and Assessment – Part III – Assessment Techniques and Earned Value Calculation", Kathryn Kulick, http://www.acq.osd.mil/pm/tpm/papers.htm

[3] "Technical Performance Measurement (TPM) Retrospective Implementation and Concept Validation on the T45TS Cockpit-21 Program", Charles Coleman, Kathryn Kulick, and CDR Nick Pisano, Published April 1996, http://www.acq.osd.mil/pm/tpm/papers.htm

APPENDIX I. Comparison of Software Life Cycle Standards

Comparison of MIL-STD-498 development activities to 12207:

MIL-STD-498 Development Activities	IEEE/EIA 12207.0 Development Activities
5.1 Project planning and oversight 5.2 Establish software devel. environment	5.3.1 Process implementation
5.3 System requirements analysis	5.3.2 System requirements analysis
5.4 System design	5.3.3 System architectural design
5.5 Software requirements analysis	5.3.4 Software requirements analysis
5.6 Software design	5.3.5 Software architectural design 5.3.6 Software detailed design
5.7 Software implementation and unit testing	5.3.7 Software coding and testing
5.8 Unit integration and testing	5.3.8 Software integration
5.9 CSCI Qualification testing	5.3.9 Software qualification testing
5.10 CSCI/HWCI integration and testing	5.3.10 Software integration
5.11 System qualification testing	5.3.11 System qualification testing
5.12 Preparing for software use	5.3.12 Software installation
5.13 Preparing for software transition	5.3.13 Software acceptance support
5.14 Software configuration management	6.2 CM Process
5.15 Software product evaluation	6.7 Audit Process
5.16 Software quality assurance	6.3 QA Process
5.17 Corrective action	6.8 Problem resolution Process
5.18 Joint technical and management reviews	6.6 Joint review Process
5.19.1 Risk management	.2 Annex L - Risk Management
5.19.2 Software management indicators	.2 Annex H - Software measurement categories
5.19.3 Security and privacy	
5.19.4 Subcontractor management	6.3.3.3 Assure prime requirements passed to subs
5.19.5 Interface with software IV&V agents	
5.19.6 Coordination with associate developers	
5.19.7 Improvement of project processes	7.3 Improvement Process

Comparison of Reviews:

DoD-STD-2167A/MIL-STD-1521B Formal Reviews	MIL-STD-498: Joint Reviews	12207.0 Joint Review Process
	Joint Technical Reviews	Technical reviews
	Joint Management Reviews	Project management reviews
	Software plan reviews	Software plan reviews
System Requirements Review (SRR)	Operational concept reviews, System/subsys reqts review	Operational concept reviews, System/subsys reqts review
System Design Review (SDR)	System/subsys design review	System/subsys design review
Software Specification Review (SSR)	Software requirements review	Software requirements review
Preliminary design Review (PDR)	Software design review	Software design review
Critical Design Review (CDR)		
Test Readiness Review (TRR)	Test readiness review Test results review Software usability review Software supportability review Critical requirements review	Test readiness review Test results review Software usability review Software supportability review Critical requirements review
Functional Configuration Audit (FCA)		
Physical Configuration Audit (PCA)		

Comparison of documents among software standards:

MIL-STD-2167 Document	, MIL-STD-498, J-STD-016 Document	MIL-STD-498 DI-IPSC-	016 Annex	12207.0 Clause	12207.1 Information Item
SDP	Software Development Plan (SDP)	81427	E.2.1	5.3.1.4 5.2.4	6.5 Development process plan 6.11 Project management plan
STP	Software Test Plan (STP)	81438	E.2.2	5.3.8	6.18 Software integration plan
IP	Software Installation Plan (SIP)	81428	E.2.3	5.5.5.2 5.3.12.1 7.4.1.1	Migration plan, Software installation plan, Training plan
CRISD	Software Transition Plan (STrP)	81429	E.2 4.	5.5.1.1 5.5.5.2 5.5.5.2 5.4	Maintenance plan, 6.8 Maintenance process plan, Migration plan 6.9 Operation process plan
SSDD	Operational Concept Description (OCD)	81430	F.2.1	5.1.1.1	6.3 Concept of operations description
SSS	System/Subsystem Spec (SSS)	81431	F.2.2	5.1.1.2 5.3.2	6.26 System requirements specification
IRS	Interface Requirements Spec (IRS)	81434	F.2.3	5.1.1.4 5.3.4	6.22 Software requirements description
SRS	Software Requirements Spec (SRS)	81433	F.2.4	5.1.1.4 5.3.4 5.3.5.5 5.3.6 5.3.7 6.5	6.22 Software requirements description, 6.27 Test or validation plan
SSDD	System/Subsys. Design Description (SSDD)	81432	G.2.1	5.3.3.1 5.3.3.2	6.25 Software arch.& reqts alloc descr
IDD	Interface Design Description (IDD)	81436	G.2.2	5.3.5.2 5.3.6.2	6.19 Software interface design descr
--	Database Design Description (DBDD)	81437	G.2.3	5.3.5.3 5.3.6.3 5.3.7.1	6.4 Database design description
SDD	Software Design Description (SDD)	81435	G.2.4	5.3.5 5.3.6	6.12 Software arch. description, 6.16 Software design description
STD	Software Test Description (STD)	81439	H.2.1	5.1.5.1 5.3.7.1 5.3.8 5.3.10 6.5	6.28 Test or validation procedures
STR	Software Test Report (STR)	81440	H.2.2	5.3.7.2 5.3.8.2 5.3.9.1 5.3.10.1 5.3.11.1 5.3.13.1 6.5	6.29 Test or validation results report
SPS	Software Product Specification (SPS)	81441	I.2.1	5.3.1.2 6.2.2.1	Software product description
VDD	Software Version Description (SVD)	81442	I.2.	6.2	6.13 Software config. index record
SPM	Computer Prog'mg Manual (CPM)	81447	I.2.3	--	--
FSM	Firmware Support Manual (FSM)	81448	I.2.4	--	--

SUM	Software User Manual (SUM)	81443	J.2.1	5.3.4.1 5.3.5.4 5.3.6.4 5.3.7.3 5.3.8.3 5.3.8.5 5.3.9.2	6.30 User documentation description
--	Software Input/Output Manual (SIOM)	81455	J.2.2	--	--
--	Software Center Operator Mnl (SCOM)	81444	J.2.3	5.4	6.9 Operation process plan
CSOM	Computer Operation Manual (COM)	81446	J.2.4	5.4	6.9 Operation process plan

APPENDIX J. COMPREHENSIVE SOFTWARE EARNED VALUE EXAMPLE

The following is a comprehensive example of performing Software earned value for a multi build software development effort. Earned value determination for selected key WBS elements at key points in the development will be discussed in order to provide insight into how earned value is applied.

NOTE - A BEFORE AND AFTER PROJECT SUMMARY ALONG WITH EVM SUMMARY AND ROLLUP DATA TABLES FOR THE EFFORT ARE INCLUDED AT THE END OF THE EXAMPLE. ALL CALCULATIONS IN THE EXAMPLE ARE BASED ON THE NON-ROUNDED NUMBERS IN THE ROLL UP TABLES. THUS IF YOU TRY TO CALCULATE CPI, SPI, ETC., BASED ON THE ROUNDED VALUES IN THE BODY OF THE EXAMPLE, THEY MAY BE SLIGHTLY DIFFERENT THAN WHAT YOU SEE IN THE EXAMPLE.

BUILD #1 EVM DISCUSSION

Build #1 consists of two CSCIs, A and B. For the purpose of this example only CSCI A will be discussed in detail. A summary of CSCI B will be provided as an input to Build #1 Integration Testing.

BEGINNING OF PROJECT MONTH 1.

1. Start Build #1, CSCI A, Software Requirements Analysis Phase.

a) 50 Systems Requirements have been allocated for implementation in Build #1, CSCI A. For the purpose of the example it will be assumed that the same amount of effort is required to decompose all systems requirements into software requirements.

b) 5 Months have been scheduled for the effort, Starting at the beginning of month 0 of the project through the end of month 5 of the project.

c) BCWS for the entire task is $1.064M or $212.8K per month.

d) Software requirements analysis of a systems requirement is considered complete when the requirements peer review is completed and all defects corrected.

END OF PROJECT MONTH 2.

1. **Build #1, CSCI A, Software Requirements Analysis Phase.**

a) Software Requirements analysis of 21 systems requirements have been completed.

b) $BCWS_{B\#1,\ CSCI\ A,\ SW\ Req} = \$425.6K$

c) $BCWP_{B\#1,\ CSCI\ A,\ SW\ Req} = (\$1.064M) \times 21/50 = \$446.9K$

d) $ACWP_{B\#1,\ CSCI\ A,\ SW\ Req} = \$436K$

e) $CPI_{B\#1,\ CSCI\ A,\ SW\ Req} = BCWP_{B\#1,\ CSCI\ A,\ SW\ Req} / ACWP_{B\#1,\ CSCI\ A,\ SW\ Req} = \$446.9K / \$436K = 1.025$

f) $SPI_{B\#1,\ CSCI\ A,\ SW\ Req} = BCWP_{B\#1,\ CSCI\ A,\ SW\ Req} / BCWS_{B\#1,\ CSCI\ A,\ SW\ Req} = \$446.9K / \$425.6K = 1.05$

BEGINNING OF PROJECT MONTH 4.

1. Start of Build #1, CSCI A, Software Design Phase.

a) Design plan based on 250 software requirements occurring as a result of software requirements analysis. Project plan will also assume approximately 30 software requirements will be added, deleted or modified during the phase due to refinement of software requirements during design. For the purpose of the example it is assumed that the same amount of effort is required to decompose all software requirements into design.

b) 12 months have been scheduled for software design, starting at the beginning of project month 4 and finishing at the end of project month 15.

c) BCWS for the entire task is $7.4784M or $623.2K per month.

d) Software design for a software requirement is considered complete when the peer review is completed and all defects corrected.

END OF PROJECT MONTH 4.

1. **Build #1, CSCI A, Software Requirements Analysis Phase.**

a) Software Requirements analysis and Peer Reviews of 41 systems requirements have been completed. However, 2 new systems requirements have been added and 3 existing ones modified, including two that had already been implemented. Since these changes are from Government ECP #1, the schedule is increased to 5.5 months, and the BCWS for the entire task is increased to $1.1704M. BCWS per month remains $212.8K.

b) $BCWS_{B\#1,\ CSCI\ A,\ SW\ Req} = \$851.2K$

c) $BCWP_{B\#1,\ CSCI\ A,\ SW\ Req} = (\$1.1704M) \times 39 / 52 = \$877.8K$

d) $ACWP_{B\#1,\ CSCI\ A,\ SW\ Req} = \$875K$

e) $CPI_{B\#1,\ CSCI\ A,\ SW\ Req} = BCWP_{B\#1,\ CSCI\ A,\ SW\ Req} / ACWP_{B\#1,\ CSCI\ A,\ SW\ Req} = \$877.8K / \$875K = 1.003$

f) $SPI_{B\#1,\ CSCI\ A,\ SW\ Req} = BCWP_{B\#1,\ CSCI\ A,\ SW\ Req} / BCWS_{B\#1,\ CSCI\ A,\ SW\ Req} = \$877.8K / \$851.2K = 1.031$

2. **Build #1, CSCI A, Software Design Phase.**

a) Software Design of 22 software requirements completed. As noted under Software requirements analysis for this month, an ECP has added 2 new systems requirements and modified 3. None of the modified systems requirements affect the design completed to date. It is now estimated there will be 260 software requirements which will need to be designed. Project schedule is increased to 12.5 months. BCWS has been increased to $7.79M, BCWS per month remains the same.

b) $BCWS_{B\#1,\ CSCI\ A,\ SW\ Des} = \$623.2K$

c) $BCWP_{B\#1,\ CSCI\ A,\ SW\ Des} = (\$7.79M) \times 22 / 260 = \$659.2K$

d) $ACWP_{B\#1,\ CSCI\ A,\ SW\ Des} = \$699K$

e) $CPI_{B\#1,\ CSCI\ A,\ SW\ Des} = BCWP_{B\#1,\ CSCI\ A,\ SW\ Des} / ACWP_{B\#1,\ CSCI\ A,\ SW\ Des} = \$659.2K / \$699K = .943$

f) $SPI_{B\#1,\ CSCI\ A,\ SW\ Des} = BCWP_{B\#1,\ CSCI\ A,\ SW\ Des} / BCWS_{B\#1,\ CSCI\ A,\ SW\ Des} = \$659.2K / \$623.2K = 1.058$

3. Build #1, CSCI A EVM Summary

a) $BCWS_{B\#1,\,CSCI\,A} = BCWS_{B\#1,\,CSCI\,A,\,SW\,Req} + BCWS_{B\#1,\,CSCI\,A,\,SW\,Des} = \$851.2K + \$623.2K$
$= \$1.4744M$

b) $BCWP_{B\#1,\,CSCI\,A} = BCWP_{B\#1,\,CSCI\,A,\,SW\,Req} + BCWP_{B\#1,\,CSCI\,A,\,SW\,Des} = \$877.8K + \$659.2K$
$= \$1.537M$

c) $ACWP_{B\#1,\,CSCI\,A} = ACWP_{B\#1,\,CSCI\,A,\,SW\,Req} + ACWP_{B\#1,\,CSCI\,A,\,SW\,Des} = \$875K + \$699K$
$= \$1.574M$

d) $CPI_{B\#1,\,CSCI\,A} = BCWP_{B\#1,\,CSCI\,A} \big/ ACWP_{B\#1,\,CSCI\,A} = \$1.537M \big/ \$1.574M = .976$

e) $SPI_{B\#1,\,CSCI\,A} = BCWP_{B\#1,\,CSCI\,A} \big/ BCWS_{B\#1,\,CSCI\,A} = \$1.537M \big/ \$1.4744M = 1.042$

f) Software design and software requirements analysis phases have been increased in length by .5 months. Since the start of software design did not change, to date the schedule has only slipped overall by .5 months.

END OF PROJECT MONTH 6.

1. **Build #1, CSCI A, Software Requirements Analysis Phase.**

a) Software Requirements analysis and Peer Reviews of 52 systems requirements have been completed. Task finished on time at the end of 5.5 months. Total number of software requirements is 260.

a) $BCWS_{B\#1,\,CSCI\,A,\,SW\,Req} = \$1.1704M$
b) $BCWP_{B\#1,\,CSCI\,A,\,SW\,Req} = 1.1704M$
c) $ACWP_{B\#1,\,CSCI\,A,\,SW\,Req} = \$1.17M$
d) $CPI_{B\#1,\,CSCI\,A,SW\,Req} = BCWP_{B\#1,\,CSCI\,A,SW\,Req} \big/ ACWP_{B\#1,\,CSCI\,A,SW\,Req} = \$1.1704M \big/ \$1.17M = 1.0$
e) $SPI_{B\#1,\,CSCI\,A,\,SW\,Req} = 1.0$

2. **Build #1, CSCI A, Software Design Phase.**

a) 63 of the total of 260 software requirements have had their design completed.

a) $BCWS_{B\#1,\,CSCI\,A,\,SW\,Des} = \$1.8696M$
b) $BCWP_{B\#1,\,CSCI\,A,SW\,Des} = (\$7.79M) \times 63 \big/ 260 = \$1.8876M$
c) $ACWP_{B\#1,\,CSCI\,A,\,SW\,Des} = \$2.0973$
d) $CPI_{B\#1,\,CSCI\,A,SW\,Des} = BCWP_{B\#1,\,CSCI\,A,SW\,Des} \big/ ACWP_{B\#1,\,CSCI\,A,SW\,Des} = \$1.8876M \big/ \$2.0973M = .9$
e) $SPI_{B\#1,\,CSCI\,A,SW\,Des} = BCWP_{B\#1,\,CSCI\,A,SW\,Des} \big/ BCWS_{B\#1,\,CSCI\,A,SW\,Des} = \$1.8876M \big/ \$1.8696M = 1.01$

FOR DETAILS ON SUBSEQUENT EVM CALCULATIONS SEE COMPREHENSIVE EXAMPLE SPREADSHEET AT END OF EXAMPLE.

3. **Build #1, CSCI A EVM Summary**

a) $BCWS_{B\#1,\,CSCI\,A} = \$3.04M$
b) $BCWP_{B\#1,\,CSCI\,A} = \$3.058M$

c) $ACWP_{B\#1, CSCI\ A} = \$3.2673M$
d) $CPI_{B\#1, CSCI\ A} = .936$
e) $SPI_{B\#1, CSCI\ A} = 1.006$

END OF PROJECT MONTH 11.

1. Build #1, CSCI A, Software Design Phase

a) The design has been completed for 165 software requirements. However 36 additions and modifications of requirements have been made to the software requirements. None of these were due to government ECPs. As a result of these changes the design of 14 software requirements must be reworked and the total number of requirements has increased to 282.
b) $BCWS_{B\#1, CSCI\ A, SW\ Des} = \$4.9856M$
c) $BCWP_{B\#1, CSCI\ A, SW\ Des} = \$4.1712M$
d) $ACWP_{B\#1, CSCI\ A, SW\ Des} = \$5.5928M$
e) $CPI_{B\#1, CSCI\ A, SW\ Des} = .746$
f) $SPI_{B\#1, CSCI\ A, SW\ Des} = .837$

2. Build #1, CSCI A EVM Summary

a) $BCSW_{B\#1, CSCI\ A} = \$6.156M$
b) $BCWP_{B\#1, CSCI\ A} = \$5.3416M$
c) $ACWP_{B\#1, CSCI\ A} = \$6.7628M$
d) $CPI_{B\#1, CSCI\ A} = .79$
e) $SPI_{B\#1, CSCI\ A} = .868$

BEGINNING OF PROJECT MONTH 12.

1. Start of Build #1, CSCI A, C&UT Phase.

a) The original C&UT plan was based on 250 software requirements occurring as a result of software requirements analysis it also assumed approximately 30 software requirements will be added, deleted or modified during the C&UT phase due to refinement of software requirements during design. For the purpose of the example it is assumed that the same amount of effort is required to decompose all software requirements into design.
b) Originally 12 months have been scheduled for C&UT, starting at the beginning of project month 12 and finishing at the end of project month 23.
c) Original BCWS for the entire task is $7.6608M or $638.4K per month.
d) Due to the ECP noted in project month 4, the schedule for this task was increased to 12.5 months and the total BCWS for the task was increased to $7.98M, BCWS per month remains the same. Since the task start date did not change, the schedule has only slipped overall by .5 months at this time. Current plan assumes there will be 260 software requirements initially which will increase to 290 by the end of C&UT.
e) The software requirements increase to 282, noted in project month 11, was not caused by a Government ECP. Thus a replan is not performed to take them into account.
f) Software C&UT for a software requirement is considered complete when the peer review is completed and all defects corrected.

END OF PROJECT MONTH 14.

1. Build #1, CSCI A, Software Design Phase

a) The design has been completed for 243 software requirements. However 19 additions and modifications of requirements have been made to the software requirements. None of these were due to government ECPs. As a result of these changes the design of 11 software requirements must be reworked and the total number of requirements has increased to 290.
b) $BCWS_{B\#1, CSCI\ A, SW\ Des} = \$6.8552M$
c) $BCWP_{B\#1, CSCI\ A, SW\ Des} = \$6.232M$
d) $ACWP_{B\#1, CSCI\ A, SW\ Des} = \$8.7112M$
e) $CPI_{B\#1, CSCI\ A, SW\ Des} = .715$
f) $SPI_{B\#1, CSCI\ A, SW\ Des} = .909$

2. Build #1, CSCI A, C&UT

a) The C&UT had been completed for 72 software requirements. Due to the software requirements changes noted in Project month 14, 1.a), the total number of requirements has increased to 290. Additionally the code for 3 software requirements will need to be reworked due to requirements modifications.
b) $BCWS_{B\#1, CSCI\ A, C\&UT} = \$1.9152M$
c) $BCWP_{B\#1, CSCI\ A, C\&UT} = \$1.8987M$
d) $ACWP_{B\#1, CSCI\ A, C\&UT} = \$1.9000M$
e) $CPI_{B\#1, CSCI\ A, C\&UT} = .999$
f) $SPI_{B\#1, CSCI\ A, C\&UT} = .991$

3. Build #1, CSCI A EVM Summary

a) $BCSW_{B\#1, CSCI\ A} = \$9.9408M$
b) $BCWP_{B\#1, CSCI\ A} = \$9.3011M$
c) $ACWP_{B\#1, CSCI\ A} = \$11.7812M$
d) $CPI_{B\#1, CSCI\ A} = .789$
e) $SPI_{B\#1, CSCI\ A} = .936$

END OF PROJECT MONTH 16.

1. Build #1, CSCI A, Software Design Phase

a) The design has been completed for all 290 software requirements.
b) $BCWS_{B\#1, CSCI\ A, SW\ Des} = \$7.79M$
c) $BCWP_{B\#1, CSCI\ A, SW\ Des} = \$7.79M$
d) $ACWP_{B\#1, CSCI\ A, SW\ Des} = \$10.7905M$
e) $CPI_{B\#1, CSCI\ A, SW\ Des} = .722$
f) $SPI_{B\#1, CSCI\ A, SW\ Des} = 1.0$

2. Build #1, CSCI A, C&UT

a) The C&UT had been completed for 117 software requirements.
b) $BCWS_{B\#1, CSCI\ A, C\&UT} = \$3.192M$
c) $BCWP_{B\#1, CSCI\ A, C\&UT} = \$3.2195M$
d) $ACWP_{B\#1, CSCI\ A, C\&UT} = \$3.1667M$
e) $CPI_{B\#1, CSCI\ A, C\&UT} = 1.017$
f) $SPI_{B\#1, CSCI\ A, C\&UT} = 1.009$

3. Build #1, CSCI A EVM Summary

a) $BCSW_{B\#1, CSCI\ A} = \$12.1524M$
b) $BCWP_{B\#1, CSCI\ A} = \$12.1799M$
c) $ACWP_{B\#1, CSCI\ A} = \$15.1271M$
d) $CPI_{B\#1, CSCI\ A} = .805$
e) $SPI_{B\#1, CSCI\ A} = 1.002$

END OF PROJECT MONTH 19.

1. Build #1, CSCI A, C&UT

a) The C&UT had been completed for 189 software requirements. ECP #2, resulting in 2 new systems requirements is implemented. It is assumed that this will result in 10 new software requirements. Due to staffing limitations, the C&UT phase schedule must be extended in order to implement these requirements. An additional 1 month will be required for requirements analysis, design and C&UT of the new requirements. Additionally, cost for the task will increase by $680.2K, total BCWS for the task will increase to $8.470.2M and BCWS per month to $646K for subsequent months. An additional 15 software requirements were added and changed due to defect correction and refinement of the requirements analysis, not through ECPs. These changes result in the code for 9 software requirements requiring revision and 6 new requirements. The end result is that the C&UT for 180 software requirements are complete and the total number of software requirements has increased to 306.

b) Additionally, due to the delay from the original schedule, the CSCI Integration testing which was planned to start at the beginning of month 20 must be delayed for two months until the beginning of month 22. This delay is justified as being a result of the two previous Government ECPs so the project is replanned to account for them.

c) $BCWS_{B\#1, CSCI\ A, C\&UT} = \$5.1072M$
d) $BCWP_{B\#1, CSCI\ A, C\&UT} = \$5.0942M$
e) $ACWP_{B\#1, CSCI\ A, C\&UT} = \$5.0667M$
f) $CPI_{B\#1, CSCI\ A, C\&UT} = 1.005$
g) $SPI_{B\#1, CSCI\ A, C\&UT} = .997$

2. Build #1, CSCI A EVM Summary

a) $BCSW_{B\#1, CSCI\ A} = \$14.0676M$
b) $BCWP_{B\#1, CSCI\ A} = \$14.0546M$
c) $ACWP_{B\#1, CSCI\ A} = \$17.0271M$
d) $CPI_{B\#1, CSCI\ A} = .825$
e) $SPI_{B\#1, CSCI\ A} = .999$

END OF PROJECT MONTH 21.

1. Build #1, CSCI A, C&UT

a) The C&UT had been completed for 226 software requirements.
b) $BCWS_{B\#1, CSCI\ A, C\&UT} = \$6.3992M$
c) $BCWP_{B\#1, CSCI\ A, C\&UT} = \$6.3961M$
d) $ACWP_{B\#1, CSCI\ A, C\&UT} = \$6.2998M$
e) $CPI_{B\#1, CSCI\ A, C\&UT} = 1.015$
f) $SPI_{B\#1, CSCI\ A, C\&UT} = 1.0$

2. Build #1, CSCI A EVM Summary

a) $BCSW_{B\#1,\ CSCI\ A}$ = $15.3596M
b) $BCWP_{B\#1,\ CSCI\ A}$ = $15.3565M
c) $ACWP_{B\#1,\ CSCI\ A}$ = $18.2603M
d) $CPI_{B\#1,\ CSCI\ A}$ = .841
e) $SPI_{B\#1,\ CSCI\ A}$ = 1.0

BEGINNING OF PROJECT MONTH 22.

1. Start of Build #1, CSCI A, CSCI Integration Testing

a) The original plan called for the CSCI Integration testing to commence during the 20th project month. As previously discussed, requirements growth to date and associated schedule slips made this impractical and the start of CSCI Integration testing was delayed by 2 months. Original assumptions were that the number of software requirements to be tested in this phase would have grown to 310, take 7 months to test and would have a BCWS for the entire task of $7.1288M. The schedule has been revised to assume the requirements will grow to 318, schedule for testing will remain 7 months, but BCWS for the entire task will increase to $7.448M or $1.0697 per month. Rework of defects found during CSCI Integration Testing is included in the estimate rework includes retesting to verify the software was correctly reworked. It is assumed that each requirement will need the same testing and rework effort for the purpose of this example.

b) The project plan calls for 95% or 302 (if there end up being 318 total software requirements) of all software requirements to pass their test procedures. The rework and testing for the remaining 5% or 16 software requirements will be deferred until build #2. If 302 software requirements do pass CSCI testing than full-earned value for the phase will be earned. This means that any software requirements affected by a priority 1 or 2 defect have failed. Additionally, there can be no more than 15 open priority 3 defects at the end of CSCI integration testing on the code for the 302 software requirements. If there are more than 15 open priority 3 defects, than the number of software requirements considered to have passed will be reduced, which will reduce the total BCWP earned during task. The remaining BCWP for the failed software requirements will not be earned until the defects are corrected in a later testing phase.

END OF PROJECT MONTH 25.

1. Build #1, CSCI A, C&UT

a) The C&UT has been completed for all 306 software requirements.
b) $BCWS_{B\#1,\ CSCI\ A,\ C\&UT}$ = $8.6602M
c) $BCWP_{B\#1,\ CSCI\ A,\ C\&UT}$ = $8.6602M
d) $ACWP_{B\#1,\ CSCI\ A,\ C\&UT}$ = $8.6M
e) $CPI_{B\#1,\ CSCI\ A,\ C\&UT}$ = 1.007
f) $SPI_{B\#1,\ CSCI\ A,\ C\&UT}$ = 1.0

2. Build #1, CSCI A, CSCI Integration Testing

a) The CSCI Integration testing had been completed for 140 of 306 software requirements. These 140 requirements are the software requirements that have successfully completed their test procedures with no priority 1 or 2 defects and no more than 1 priority 3 defect for every 20 software requirements. In this case test procedures have actually been executed on 212

software requirements, but 62 of them have priority 1 or 2 defects, or an excessive number of priority 3 defects. Since the testing was planned for 318 software requirements, in order to be on schedule for successfully testing 95% of them, or 302, 173 should have successful completed testing by this time. With 306 total requirements, at a minimum 166 should have been successfully tested at this point, so they are still behind even for the smaller number of requirements currently existing.

b) $BCWS_{B\#1,\ CSCI\ A,\ Int\ tes} = \$4.256M$
c) $BCWP_{B\#1,\ CSCI\ A,\ Int\ tes} = \$3.5956M$
d) $ACWP_{B\#1,\ CSCI\ A,\ Int\ tes} = \$4.3M$
e) $CPI_{B\#1,\ CSCI\ A,\ Int\ tes} = 0.836$
f) $SPI_{B\#1,\ CSCI\ A,\ Int\ tes} = 0.845$

3. Build #1, CSCI A EVM Summary

a) $BCSW_{B\#1,\ CSCI\ A} = \$21.8766M$
b) $BCWP_{B\#1,\ CSCI\ A} = \$21.2162M$
c) $ACWP_{B\#1,\ CSCI\ A} = \$24.8605M$
d) $CPI_{B\#1,\ CSCI\ A} = .853$
e) $SPI_{B\#1,\ CSCI\ A} = .97$

END OF PROJECT MONTH 28.

1. Build #1, CSCI A, CSCI Integration Testing

a) The CSCI Integration testing had been completed for 245 of 306 software requirements. Testing was scheduled to be completed at this point, however to earn 100% of the available earned value, 95% or 290 of the software requirements must have been successfully tested. If testing stopped at this point, the remaining earned value could not be earned until the testing for these software requirements were successfully completed. In our example we will assume that testing must continue due to the critical nature of the remaining 45 software requirements, which must be tested. Additionally, Build #1 Integration Testing, which would perform Integration testing of the two CSCIs in Build #1, CSCIs A & B, was scheduled to start at the beginning of project month 29 after the CSCI Integration testing for Builds A and B had completed. This must now be delayed until Build A CSCI Integration testing can be completed.

b) $BCWS_{B\#1,\ CSCI\ A,\ Int\ test} = \$7.448M$
c) $BCWP_{B\#1,\ CSCI\ A,\ Int\ tes} = \$6.2923M$
d) $ACWP_{B\#1,\ CSCI\ A,\ Int\ tes} = \$7.556M$
e) $CPI_{B\#1,\ CSCI\ A,\ Int\ tes} = 0.833$
f) $SPI_{B\#1,\ CSCI\ A,\ Int\ tes} = 0.845$

2. Build #1, CSCI A EVM Summary

a) $BCSW_{B\#1,\ CSCI\ A} = \$25.0686M$
b) $BCWP_{B\#1,\ CSCI\ A} = \$23.9129M$
c) $ACWP_{B\#1,\ CSCI\ A} = \$28.1165M$
d) $CPI_{B\#1,\ CSCI\ A} = .85$
e) $SPI_{B\#1,\ CSCI\ A} = .954$

MIDWAY THROUGH PROJECT MONTH 30.

1. Start of Build #1 Integration testing.

a) For the purpose of the example, a detailed EVM example for Build #1, CSCI B will not be provided. Build B's EVM situation will be assumed to be identical to that of CSCI A at the beginning of Build #1 Integration testing. Including the total current number of Software requirements.

b) The original project plan called for Build #1 Integration Testing to start at the beginning of the 27th month. The last replan following the ECP #2 discussed in the 19th project month called for Build #1 Integration Testing to start at the beginning of the 29th month. Due to problems during CSCI Integration testing the start was delayed until the middle of the 30th month. The project was not replanned due to this issue, which means for the first month and a half of Build #1 integration testing no BCWP was earned for the effort.

c) The Integration Testing was planned to take 5 months with a total BCWS for the task of $10.64M, or $2.128M per month. Project plan called for 636 software requirements to be tested, 318 in each CSCI. Rework of defects found during CSCI Integration Testing is included in the estimate rework includes retesting to verify the software was correctly reworked. It is assumed that each requirement will need the same testing and rework effort for the purpose of this example.

d) The project plan calls for 95% or 604 (if there end up being 636 total software requirements) of all software requirements to pass their test procedures. The rework and testing for the remaining 5% or 32 software requirements will be deferred until build #2. If 604 software requirements do pass CSCI testing than full-earned value for the phase will be earned. Any software requirements affected by a priority 1 or 2 defect have failed. Additionally, there can be no more than 30 open priority 3 defects at the end of Build #1 integration testing on the code for the 504 software requirements. If there are more than 15 open priority 3 defects, than the number of software requirements considered to have passed will be reduced, which will reduce the total BCWP earned during task. The remaining BCWP for the failed software requirements will not be earned until the defects are corrected in a later testing phase.

END OF PROJECT MONTH 30.

1. Build #1, CSCI A, CSCI Integration Testing

a) The CSCI Integration testing had been completed for 290 of 306 software requirements after 8.5 months. This meets the project plan of having successfully tested 95% of the software requirements during this phase.

b) $BCWS_{B\#1,\ CSCI\ A,\ Int\ tes} = \$7.448M$

c) $BCWP_{B\#1,\ CSCI\ A,\ Int\ tes} = \$7.448M$

d) $ACWP_{B\#1,\ CSCI\ A,\ Int\ tes} = \$8.945M$

e) $CPI_{B\#1,\ CSCI\ A,\ Int\ tes} = 0.833$

f) $SPI_{B\#1,\ CSCI\ A,\ Int\ tes} = 1.0$

2. Build #1, CSCI A EVM Summary

a) $BCSW_{B\#1,\ CSCI\ A} = \$25.0686M$

b) $BCWP_{B\#1,\ CSCI\ A} = \$25.0686M$

c) $ACWP_{B\#1,\ CSCI\ A} = \$29.5055M$

d) $CPI_{B\#1,\ CSCI\ A} = .85$

e) $SPI_{B\#1,\ CSCI\ A} = 1.0$

3. Build #1, CSCI B EVM Summary

a) $BCSW_{B\#1,\ CSCI\ B}$ = $25.0686M
b) $BCWP_{B\#1,\ CSCI\ B}$ = $25.0686M
c) $ACWP_{B\#1,\ CSCI\ B}$ = $29.5055M
d) $CPI_{B\#1,\ CSCI\ B}$ = .85
e) $SPI_{B\#1,\ CSCI\ B}$ = 1.0

4. Build #1 Integration Testing

a) At this time 51 of 612 software requirements have been successfully tested.
b) $BCWS_{B\#1,\ Int\ tes}$ = $4.256M
c) $BCWP_{B\#1,\ Int\ tes}$ = $0.9340M
d) $ACWP_{B\#1,\ Int\ tes}$ = $1.336M
e) $CPI_{B\#1,\ Int\ tes}$ = 0.699
f) $SPI_{B\#1,\ Int\ tes}$ = 0.219

5. Build #1 Summary

a) $BCSW_{B\#1}$ = $54.3932M
b) $BCWP_{B\#1,\ CSCI\ B}$ = $51.0712M
c) $ACWP_{B\#1,\ CSCI\ B}$ = $60.3465M
d) $CPI_{B\#1,\ CSCI\ B}$ = .846
e) $SPI_{B\#1,\ CSCI\ B}$ = .939

END OF PROJECT MONTH 33.

1. Build #1 Integration Testing

a) At this time 350 of 612 software requirements have been successfully tested. The plan was to have completed testing by this time. However due to a delayed start and lower than expected progress, the task is well behind schedule. The slower than expected progress being experienced during testing indicates one off two things. The original plans were not realistic, or there is a quality problem in the development resulting in more defects being injected in earlier phases. At this point it appears that that it will take close to three months to finish the remainder of the testing.
b) $BCWS_{B\#1,\ Int\ tes}$ = $10.64M
c) $BCWP_{B\#1,\ Int\ tes}$ = $6.4096M
d) $ACWP_{B\#1,\ Int\ tes}$ = $7.498M
e) $CPI_{B\#1,\ Int\ tes}$ = 0.855
f) $SPI_{B\#1,\ Int\ tes}$ = 0.602

2. Build #1 Summary

a) $BCSW_{B\#1}$ = $60.7772M
b) $BCWP_{B\#1,\ CSCI\ B}$ = $56.5468M
c) $ACWP_{B\#1,\ CSCI\ B}$ = $66.5089M
d) $CPI_{B\#1,\ CSCI\ B}$ = .85
e) $SPI_{B\#1,\ CSCI\ B}$ = .93

END OF PROJECT MONTH 36.

1. Build #1 Integration Testing

a) At this time 581 of 612 software requirements have been successfully tested. This meets the original project plan of successfully testing 95% of the software requirements during the phase. However there has been a 3-month schedule slip. Partially due to a late start and partially due to slower than expected progress during testing.

b) $BCWS_{B\#1,\ Int\ test} = \$10.64M$

c) $BCWP_{B\#1,\ Int\ test} = \$10.64M$

d) $ACWP_{B\#1,\ Int\ test} = \$12.473$

e) $CPI_{B\#1,\ Int\ test} = 0.853$

f) $SPI_{B\#1,\ Int\ test} = 1.00$

2. Build #1 Summary

a) $BCSW_{B\#1} = \$60.7772M$

b) $BCWP_{B\#1,\ CSCI\ B} = \$60.7772M$

c) $ACWP_{B\#1,\ CSCI\ B} = \$71.4839$

d) $CPI_{B\#1,\ CSCI\ B} = .85$

e) $SPI_{B\#1,\ CSCI\ B} = .1.00$

BUILD #2 EVM DISCUSSION

Build #2 consists of two CSCIs, A and B. For the purpose of this example only CSCI A will be discussed in detail. A summary of CSCI B will be provided as an input to Build #2 Integration Testing. These are the same CSCIs discussed in Build #1, Build #2 will add additional functionality to these CSCIs and complete their testing prior to release. Build #2 will also finish any rework deferred from Build #1.

The original project plan calls for Build #2 CSCI A C&UT to start at the end of Build #1 Integration testing. Since in the original plan, Build #1 Integration testing was supposed to end at the end of the 31st project month, this means Build #2, CSCI A Software requirements Analysis should start at the beginning of the 24th project month.

END OF PROJECT MONTH 19.

1. ECP Impacts.

a) Two ECPs have occurred at this time, #1 and #2. These ECPs impacts on Build #1 CSCI A were discussed in Build #1 project months 4 and 19 discussions previously. However they have also had an impact on Build #2 CSCI A system requirements. The end result of these ECPs has been to increase the total number of systems requirements for Build #2, CSCI A to 46 from the original 40 systems requirements. Additionally, ECP #3, which affects only Build #2, is expected to be approved in the next couple of months. Assuming this occurs, the total number of systems requirements to be implemented by Build #2 CSCI A will increase to 48. Additionally, Build # 1 Integration testing is now not scheduled to end until the end of the 33rd project month due to these ECP impacts. Based on this information it is decided to revise the start date for the Build #2 CSCI A to the beginning of the 25th project month.

BEGINNING OF PROJECT MONTH 25.

1. Start Build #2, CSCI A, Software Requirements Analysis Phase.

a) At the beginning of the project 40 systems requirements had been allocated for implementation in Build #2, CSCI A.

b) In the original project plan 4 months had been scheduled for the effort, starting at the beginning of the 24th project month and extending to the end of the 27th project month. BCWS for the entire task is $1.0336M or $258.4K per month.

c) During the 19th month the project schedule for Build #2 CSCI A was revised based on ECPs #1, and #2. Since that time ECP #3 has been approved and the three of them have resulted in an increase to 48 systems requirements for Build #2 CSCI A. In the replan based on these ECPs, the start of the software requirements phase was slipped to the beginning of the 25th project month, and will end four months later at the end of the 28th project month. BCWS for the entire task has increased to $1.216M or $304K per month.

d) It is expected that the current 48 systems requirements will decompose into approximately 240 software requirements.

e) For the purpose of the example it will be assumed that the same amount of effort is required to decompose all systems requirements into software requirements.

BEGINNING OF PROJECT MONTH 27.

1. Start Build #2, CSCI A, Software Design Phase.

a) The original project plan called for the Build #2 CSCI A Software Design Phase to start at the beginning of the 26th project month. The task was scheduled to last 9 months and be completed at the end of the 34th project month. This included time for rework of designs for requirements that were deferred from Build #1 CSCI A. BCWS for the entire task was $6.688M or $743.1K per month.

b) As a result of the replan in the 19th project month based on ECPs #1, #2 and #3, the start of the task was changed to the beginning of the 27th month. The schedule was increased to 10 months resulting in a planned finish at the end of the 36th project month. The revised project plan was based on their being approximately 279 software requirements by the end of the phase, including requirements deferred from Build #1 CSCI A for rework. The total revised BCWS for the task is $7.904M or $790.4K per month.

END OF PROJECT MONTH 27.

1. Build #2, CSCI A, Software Requirements Analysis Phase.

a) Requirements analysis has been completed for 33 systems requirements, which have generated at total of 181 software requirements. Based on this it is now expected that the total number of software requirements will increase to 264 from the originally predicted 240. Task is behind schedule and over cost.

b) $BCWS_{\text{Build \#2, CSCI A, SW Req Anal}} = \$912K$

c) $BCWP_{\text{Build \#2, CSCI A, SW Req Anal}} = \$836K$

d) $ACWP_{\text{Build \#2, CSCI A, SW Req Anal}} = \$905K$

e) $CPI_{\text{Build \#2, CSCI A, SW Req Anal}} = .924$

f) $SPI_{\text{Build \#2, CSCI A, SW Req Anal}} = .917$

2. Build #2, CSCI A, Software Design Phase.

a) At this time design has been completed for 28 software requirements. However the total number of software requirements expected to be generated out of the Build #2 CSCI A

software requirements analysis phase has increased to 264. Taking into account further requirements growth during the design phase and reword of deferred Build #1 CSCI A requirements it is now predicted there will be a total of 306 software requirements that must be designed by the end of the design phase. The number of requirements to be deferred from Build #1 CSCI A is still unknown at this point due to testing being incomplete. Earned value will be determined based on the actual current number of known requirements.

b) $BCWS_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$790.4K$
c) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$838.3K$
d) $ACWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$795K$
e) $CPI_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = 1.054$
f) $SPI_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = 1.061$

3. Build #2 CSCI A Summary

a) $BCWP_{Build\ \#2,\ CSCI\ A} = \$1.7024M$
b) $BCWP_{Build\ \#2,\ CSCI\ A} = \$1,6743M$
c) $ACWP_{Build\ \#2,\ CSCI\ A} = \$1.7M$
d) $CPI_{Build\ \#2,\ CSCI\ A} = .985$
e) $SPI_{Build\ \#2,\ CSCI\ A} = .983$

END OF PROJECT MONTH 28.

1. Build #2, CSCI A, Software Requirements Analysis Phase.

a) Requirements analysis has been completed for 45 systems requirements, which have generated at total of 247 software requirements. Total final estimate of software requirements remains 264. Task was expected to be completed at this time, there appears to be at least a week or more work remaining.

b) $BCWS_{Build\ \#2,\ CSCI\ A,\ SW\ Req\ Anal} = \$1.216M$
c) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Req\ Anal} = \$1.140M$
d) $ACWP_{Build\ \#2,\ CSCI\ A,\ SW\ Req\ Anal} = \$1.199$
e) $CPI_{Build\ \#2,\ CSCI\ A,\ SW\ Req\ Anal} = .951$
f) $SPI_{Build\ \#2,\ CSCI\ A,\ SW\ Req\ Anal} = .938$

2. Build #2, CSCI A, Software Design Phase.

a) At this time design has been completed for 55 software requirements.
b) $BCWS_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$1.5808M$
c) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$1.6467M$
d) $ACWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$1.59M$
e) $CPI_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = 1.036$
f) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = 1.042$

3. Build #2 CSCI A Summary

a) $BCWP_{Build\ \#2,\ CSCI\ A} = \$2.7968M$
b) $BCWP_{Build\ \#2,\ CSCI\ A} = \$2.7867M$
c) $ACWP_{Build\ \#2,\ CSCI\ A} = \$2.789M$
d) $CPI_{Build\ \#2,\ CSCI\ A} = .999$
e) $SPI_{Build\ \#2,\ CSCI\ A} = .996$

END OF PROJECT MONTH 29.

1. Build #2, CSCI A, Software Requirements Analysis Phase.

a) Requirements analysis has been completed for all 48 systems requirements, which have generated at total of 270 software requirements. Task finished approximately a week behind schedule.

b) $BCWS_{Build\ \#2,\ CSCI\ A,\ SW\ Req\ Anal} = \$1.216M$

c) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Req\ Anal} = \$1.216M$

d) $ACWP_{Build\ \#2,\ CSCI\ A,\ SW\ Req\ Anal} = \$1.280$

e) $CPI_{Build\ \#2,\ CSCI\ A,\ SW\ Req\ Anal} = .95$

f) $SPI_{Build\ \#2,\ CSCI\ A,\ SW\ Req\ Anal} = 1.0$

2. Build #2, CSCI A, Software Design Phase.

a) At this time design has been completed for 85 software requirements of the current total of 270.

b) $BCWS_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$2.3712M$

c) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$2.4883M$

d) $ACWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$2.411M$

e) $CPI_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = 1.032$

f) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = 1.049$

3. Build #2 CSCI A Summary

a) $BCWP_{Build\ \#2,\ CSCI\ A} = \$3.5872M$

b) $BCWP_{Build\ \#2,\ CSCI\ A} = \$3.7043M$

c) $ACWP_{Build\ \#2,\ CSCI\ A} = \$3.691M$

d) $CPI_{Build\ \#2,\ CSCI\ A} = 1.004$

e) $SPI_{Build\ \#2,\ CSCI\ A} = 1.033$

END OF PROJECT MONTH 33.

1. Build #2, CSCI A, Software Design Phase.

a) At this time design has been completed for 196 software requirements. Build #1 CSCI A CSCI Integration testing was finished in the 30^{th} month with 16 software requirements deferred to Build #2 CSCI A. Of these 16 requirements 12 require rework of their design. This design rework will require additional rework of eight of the software requirements already designed in this phase. Further an additional 24 new software requirements are necessary as a result of further software requirements analysis occurring during design. The total number of requirements has increased to 300 software requirements. Additionally, the software requirements design has been completed for has dropped to 188 do to the rework required because of those requirements deferred from build #1 CSCI A.

b) The project plan had called for starting Build #2, CSCI A C&UT at the beginning of the 34^{th} project month after the completion of Build #1 Integration Testing. However, it now appears that it will take another 3 months before Build #1 Integration testing will be completed. Further, Build #2 CSCI A software design is running behind schedule due to the higher than expected number of software requirements. It now appears that Build #2 CSCI A software design will take an extra month to complete. C&UT will therefore be delayed until the beginning of the 35^{th} project month, the additional risk associated with starting C&UT prior to completing Build #1 Integration Testing will be accepted and monitored in order to

avoid loss of additional schedule. Since these delays were not caused by Government ECPs, a replan will not be conducted.

c) $BCWS_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$5.5328M$
d) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$4.9532M$
e) $ACWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$5.809M$
f) $CPI_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = .853$
g) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = .895$

2. Build #2 CSCI A Summary

a) $BCWP_{Build\ \#2,\ CSCI\ A} = \$6.7488M$
b) $BCWP_{Build\ \#2,\ CSCI\ A} = \$6.1692M$
c) $ACWP_{Build\ \#2,\ CSCI\ A} = \$7.089M$
d) $CPI_{Build\ \#2,\ CSCI\ A} = .87$
e) $SPI_{Build\ \#2,\ CSCI\ A} = .914$

BEGINNING OF PROJECT MONTH 35.

1. Start Build #2, CSCI A, C&UT Phase.

a) The original plan called for C&UT to begin at the beginning of the 34th project month. The effort was scheduled to last 9 months, completing at the end of the 42nd project month. Schedule and resources were based on the assumption that there would be approximately 279 software requirements including deferred requirements from Build #1 CSCI A to be coded during the phase. It was further assumed that 25 software requirements would be added or modified by the end of the phase. Schedule and personnel to implement these requirements was sized accordingly. BCWS for the entire task was $9.179M or $1.02M per month.

b) Due to delays in software design for Build #2 CSCI A and Build #1 Integration testing the start of Build #2 CSCI C&UT was delayed until the beginning of the 35th project month. The base project plan has not been replanned or rebaselined to insure EVM continues to provide useful information. Additionally these delays were not caused by an ECP so rebaselining to account for increased scope is inappropriate. This means that no BCWS or ACWP will accumulate the during the 34th project month when this task was supposed to have started. This means a likely slip of one month on the end of the effort. There are currently 300 software requirements expected to complete the Build #2 CSCI A software design phase. This included 12 software requirements deferred from build #1 CSCI A. There are also an additional 4 software requirements that were deferred from Build #1, CSCI A which while they didn't need design rework, do need to be reworked during the Build #2, CSCI A C&UT phase. Thus there are a total of 304 software requirements. The task was originally planned based on starting with 279 with 25 additions and changes, almost all of the preplanned growth has already been used up. If requirements growth continues at previous rates this will result in cost overruns and probably additional schedule slips.

END OF PROJECT MONTH 37.

1. Build #2, CSCI A, Software Design Phase.

a) Software design for all 300 requirements was completed 1 month late.
b) $BCWS_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$7.904$
c) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$7.904$
d) $ACWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = \$9.128$

e) $CPI_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = .866$
f) $BCWP_{Build\ \#2,\ CSCI\ A,\ SW\ Des} = 1.0$

2. Build #2 CSCI A, C&UT Phase

a) C&UT for 111 of 304 software requirements have been completed. As a result of the completion of Build #1, Integration testing 15 software requirements were deferred for additional rework in Build #2 CSCI A. Of these 15, 10 were determined to have been related to issues previously identified and deferred from Build #1, CSCI A CSCI Integration testing and are already included in the Build #2 CSCI A, C&UT implementation. The remaining 5 deferred software requirements are additional tasking for this phase, resulting in the total number of software requirements increasing to 309. Additionally, 4 other software requirements have been changed and 4 new software requirements have been added. Bringing the total number of software requirements to 313. An ECP is also currently in work.
b) $BCWS_{Build\ \#2,\ CSCI\ A,\ C\&UT} = \$4.0799M$
c) $BCWP_{Build\ \#2,\ CSCI\ A,\ C\&UT} = \$3.2554M$
d) $ACWP_{Build\ \#2,\ CSCI\ A,\ C\&UT} = \$3.047$
e) $CPI_{Build\ \#2,\ CSCI\ A,\ C\&UT} = 1.068$
f) $BCWP_{Build\ \#2,\ CSCI\ A,\ C\&UT} = .798$

3. Build #2 CSCI A Summary

a) $BCWP_{Build\ \#2,\ CSCI\ A} = \$13.1999M$
b) $BCWP_{Build\ \#2,\ CSCI\ A} = \$12.3754M$
c) $ACWP_{Build\ \#2,\ CSCI\ A} = \$13.455M$
d) $CPI_{Build\ \#2,\ CSCI\ A} = .92$
e) $SPI_{Build\ \#2,\ CSCI\ A} = .938$

END OF PROJECT MONTH 40.

1. Build #2 CSCI A, C&UT Phase

a) C&UT for 203 of 313 software requirements have been completed. Based on this data, it appears that the C&UT phase will extend at least into the beginning of the 44^{th} project month.
b) ECP #4 was approved and will be considered by EVM in all future months. This ECP adds 5 new systems requirements. This will increase BCWP for the task by \$1.7539M to \$10.9336M. BCWS per month for all subsequent months of the task will be \$948.5K. Schedule will increase by 2 months from 9 months to 11 months.
c) The project plan amended for ECPs #1, 2 and 3, called for Build #2 CSCI A, CSCI Integration Testing to commence at the beginning of the 41^{st} month. As a result of ECP #4, this will be delayed until the beginning of the 43^{rd} month.
d) $BCWS_{Build\ \#2,\ CSCI\ A,\ C\&UT} = \$7.1398M$
e) $BCWP_{Build\ \#2,\ CSCI\ A,\ C\&UT} = \$5.9536M$
f) $ACWP_{Build\ \#2,\ CSCI\ A,\ C\&UT} = \$5.8729M$
g) $CPI_{Build\ \#2,\ CSCI\ A,\ C\&UT} = 1.014$
h) $BCWP_{Build\ \#2,\ CSCI\ A,\ C\&UT} = .834$

2. Build #2 CSCI A Summary

a) $BCWS_{Build\ \#2,\ CSCI\ A} = \$16.2598M$
b) $BCWP_{Build\ \#2,\ CSCI\ A} = \$15.0736M$

c) $ACWP_{Build\ \#2,\ CSCI\ A}$ = $16.2809M
d) $CPI_{Build\ \#2,\ CSCI\ A}$ = .926
e) $SPI_{Build\ \#2,\ CSCI\ A}$ = .927

END OF PROJECT MONTH 42.

1. Build #2 CSCI A, C&UT Phase

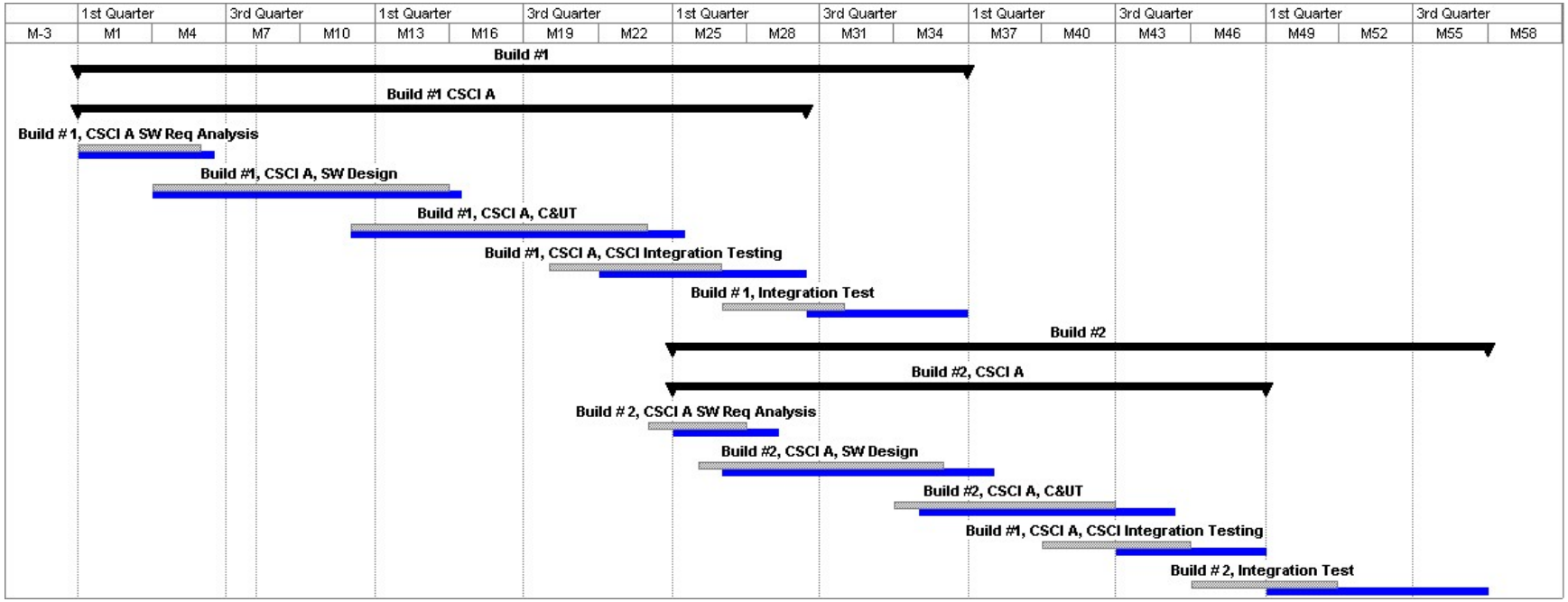
a) As a result of the software requirements analysis of the 5 new systems requirements in ECP #4, 20 new software requirements were generated and the design and code for 8 existing requirements were revised. At this point there are 333 software requirements of which 263 have been successfully completed C&UT.
b) $BCWS_{Build\ \#2,\ CSCI\ A,\ C\&UT}$ = $9.0367M
c) $BCWP_{Build\ \#2,\ CSCI\ A,\ C\&UT}$ = $7.7133M
d) $ACWP_{Build\ \#2,\ CSCI\ A,\ C\&UT}$ = $8.2322M
e) $CPI_{Build\ \#2,\ CSCI\ A,\ C\&UT}$ = .937
f) $BCWP_{Build\ \#2,\ CSCI\ A,\ C\&UT}$ = .854

2. Build #2 CSCI A Summary

a) $BCWS_{Build\ \#2,\ CSCI\ A}$ = $18.1567M
b) $BCWP_{Build\ \#2,\ CSCI\ A}$ = $16.8333M
c) $ACWP_{Build\ \#2,\ CSCI\ A}$ = $18.6402M
d) $CPI_{Build\ \#2,\ CSCI\ A}$ = .903
e) $SPI_{Build\ \#2,\ CSCI\ A}$ = .927

Discussion of the remaining test phases will be skipped since they provide no additional insight not provided in previous discussion. The remaining phases are included in the following schedule and EVM summary tables.

Project Schedule



Project Schedule Key

- ✓ Gray - Initial Project Schedule.
- ✓ Blue – Actual Project Schedule. Schedule caused by approved ECPs and those caused by overruns of initial project schedule are not differentiated.

Project Schedule Notes

- ✓ While each build includes CSCI A and a CSCI B, CSCI B is not shown on the schedule, just as it is not discussed in the example.
- ✓ While CSCI B is not included in the schedule, its integration with CSCI A is considered to be included in Build #1 Integration Test and Build #2 Integration Test.

Comprehensive Example EVM Summary Table Interpretation

Interpretation:

- ✓ Phase Table – A task table shows EVM data for specific project phase.
- ✓ Summary Table – Shows summary or rollup data from several phases. CSCI Summary tables roll up data from the: software requirements analysis, software design, C&UT, and CSCI Integration Testing phases for the current build. Build Summary tables roll up data from all phases and CSCIs in the build. The Project Summary Table rolls up data from all phases in all builds and CSCIs of the project.
- ✓ Project Month column – Current month of the project for which the EVM data in the associated row was collected. Month 1 is the 1st month of the project. EVM data is always as of the end of the project month.
- ✓ Total BCWS column – Total BCWS allocated for a particular task. In a summary table it is the total BCWS for all started and completed tasks to date. Changes in the Total BCWS from one month to another indicate that an ECP has been approved and the project budget revised. In the discussion for the example, in some cases, the initial Total BCWS is different than the values in the tables. This occurs when the initial budget has been revised prior to the start of the task. Only the Total BCWS at the time the task is currently scheduled to start and any changes after the currently scheduled start will appear in the phase table.
- ✓ Schedule Length in months column– Specifies the planned length of the phase in months. Appears only in phase tables. Changes in the length from month to month indicate an ECP has been approved which changed the project schedule.
- ✓ Total Requirements column – The total number of requirements, software or system, which are planned for implementation in the phase. Appears only in phase tables. Changes in the total number of requirements can occur due to ECPs, or due to non-contractual changes in the number of requirements to be implemented.
- ✓ System or Software Requirements Completed to Data column – Total number of requirements for which peer reviews in the phase have been successfully completed in the current project month. Appears only in phase tables.
- ✓ BCWS column – In phase tables, identifies BCWS up to and including the current project month. In Summary table rollups, it includes the BCWS for all summarized tasks started and completed prior to or during the current project month.
- ✓ BCWP column – In phase tables, identifies BCWP up to and including the current project month. In Summary table rollups, it includes the BCWP for all summarized tasks started and completed prior to or during the current project month.
- ✓ ACWP column – In phase tables, identifies ACWP up to and including the current project month. In Summary table rollups, it includes the ACWP for all summarized tasks started and completed prior to or during the current project month.
- ✓ CPI column – Self explanatory.
- ✓ SPI column – Self explanatory.
- ✓ Start Month column – Project month during which the phase is planned to begin. Appears only in phase tables. In a few cases in the textual description of the project, it specifies that the phase was originally planned to start prior to the planned start month.

This indicates that an ECP was approved prior to the start of the phase, which resulted in a replan of the project schedule. Once a phase starts, this column does not change.

- ✓ End Month Column – Project month during which the phase was planned to be completed. Only appears in phase tables. If the end month changes, it indicates that an ECP was approved resulting in

Comprehensive Example EVM Summary

Build #1, CSCI A, SW Requirements Analysis

Project Month	Total BCWS	Schedule length in months	Total requirements	System Requirements Completed to date	BCWS	BCWP	ACWP	CPI	SPI	Start Month	End Month
2	$1,064,000	5	50	21	$425,600	$446,880	$436,000	1.025	1.050	0	5
4	$1,170,400	5.5	52	39	$851,200	$877,800	$875,000	1.003	1.031	0	6
6	$1,170,400	5.5	52	52	$1,170,400	$1,170,400	$1,170,000	1.000	1.000	0	6

Original Schedule: Start month 0, End Month 5, BCWS = $1,064,000
Final Schedule: Start Month 0, End Month 5.5, BCWS = $1,170,400
Schedule extended and BCWS revised due to ECP #1 in Month 4

Build #1, CSCI A, SW Design

Project Month	Total BCWS	Schedule length in months	Total requirements	Software Requirements Completed to date	BCWS	BCWP	ACWP	CPI	SPI	Start Month	End Month
4	$7,790,000	12.5	260	22	$623,200	$659,154	$699,000	0.943	1.058	4	16
6	$7,790,000	12.5	260	63	$1,869,600	$1,887,577	$2,097,308	0.900	1.010	4	16
11	$7,790,000	12.5	282	151	$4,985,600	$4,171,241	$5,592,821	0.746	0.837	4	16
14	$7,790,000	12.5	290	232	$6,855,200	$6,232,000	$8,711,242	0.715	0.909	4	16
16	$7,790,000	12.5	290	290	$7,790,000	$7,790,000	$10,790,468	0.722	1.000	4	16

Original Schedule: Start month 4, End Month 15, BCWS = $7,478,400
Final Schedule, Start Month 4, End Month 15.5, BCWS $7,790,000
Schedule Extended and BCWS revised due to ECP #1 in month 4.

Build #1, CSCI A, C&UT

Project Month	Total BCWS	Schedule length in months	Total requirements	Software Requirements Completed to date	BCWS	BCWP	ACWP	CPI	SPI	Start Month	End Month
14	$7,980,000	12.5	290	69	$1,915,200	$1,898,690	$1,900,000	0.999	0.991	12	24
16	$7,980,000	12.5	290	117	$3,192,000	$3,219,517	$3,166,677	1.017	1.009	12	24
19	$8,660,200	13.5	306	180	$5,107,200	$5,094,235	$5,066,683	1.005	0.997	12	25
21	$8,660,200	13.5	306	226	$6,399,200	$6,396,095	$6,299,844	1.015	1.000	12	25
25	$8,660,200	13.5	306	306	$8,660,200	$8,660,200	$8,600,000	1.007	1.000	12	25

Original Schedule: Start Month 12, End Month 23, BCWS = $7,660,800
Final Schedule: Start Month 12, End Month , BCWS = $8,660,200
Schedule Extended and BCWS revised due to ECP #1 in month 4 and ECP #2 in month 19.

Build #1, CSCI A, CSCI Integration Test

Project Month	Total BCWS	Schedule length in months	Total requirements	Software Requirements Completed to date	BCWS	BCWP	ACWP	CPI	SPI	Start Month	End Month
25	$7,448,000	7	306	140	$4,256,000	$3,595,586	$4,300,000	0.836	0.845	22	28
28	$7,448,000	7	306	245	$7,448,000	$6,292,276	$7,556,000	0.833	0.845	22	28
30	$7,448,000	8.5	306	290	$7,448,000	$7,448,000	$8,945,000	0.833	1.000	22	30

Original Schedule: Start Month 20, End Month 26, BCWS = $7,128,800
Final Schedule: Start Month 22, End Month 30, BCWS = $7,448,000
Schedule Extended and BCWS revised due to ECP #1 in month 4 and ECP #2 in month 19.

Build #1, CSCI A Summary

Project Month	Total BCWS	BCWS	BCWP	ACWP	CPI	SPI
2	$1,064,000	$425,600	$446,880	$436,000	1.025	1.050
4	$8,960,400	$1,474,400	$1,536,954	$1,574,000	0.976	1.042
6	$8,960,400	$3,040,000	$3,057,977	$3,267,308	0.936	1.006
11	$8,960,400	$6,156,000	$5,341,641	$6,762,821	0.790	0.868
14	$16,940,400	$9,940,800	$9,301,090	$11,781,242	0.789	0.936
16	$16,940,400	$12,152,400	$12,179,917	$15,127,145	0.805	1.002
19	$17,620,600	$14,067,600	$14,054,635	$17,027,151	0.825	0.999
21	$17,620,600	$15,359,600	$15,356,495	$18,260,312	0.841	1.000
25	$25,068,600	$21,876,600	$21,216,186	$24,860,468	0.853	0.970
28	$25,068,600	$25,068,600	$23,912,876	$28,116,468	0.850	0.954
30	$25,068,600	$25,068,600	$25,068,600	$29,505,468	0.850	1.000

Original Schedule: Start Month 0, End Month 26, BCWS = $23,332,000
Final Schedule: Start month 0, End Month 30, BCWS = $25,068,600
Schedule Extended and BCWS revised due to ECP #1 in month 4 and ECP #2 in month 19.

Build #1, CSCI B Summary

Project Month	Total BCWS	BCWS	BCWP	ACWP	CPI	SPI
2	$1,064,000	$425,600	$446,880	$436,000	1.025	1.050
4	$8,960,400	$1,474,400	$1,536,954	$1,574,000	0.976	1.042
6	$8,960,400	$3,040,000	$3,057,977	$3,267,308	0.936	1.006
11	$8,960,400	$6,156,000	$5,341,641	$6,762,821	0.790	0.868
14	$16,940,400	$9,940,800	$9,301,090	$11,781,242	0.789	0.936
16	$16,940,400	$12,152,400	$12,179,917	$15,127,145	0.805	1.002
19	$17,620,600	$14,067,600	$14,054,635	$17,027,151	0.825	0.999
21	$17,620,600	$15,359,600	$15,356,495	$18,260,312	0.841	1.000
25	$25,068,600	$21,876,600	$21,216,186	$24,860,468	0.853	0.970
28	$25,068,600	$25,068,600	$23,912,876	$28,116,468	0.850	0.954
30	$25,068,600	$25,068,600	$25,068,600	$29,505,468	0.850	1.000

Original Schedule: Start Month 0, End Month 26, BCWS = $23,332,000
Final Schedule: Start month 0, End Month 30, BCWS = $25,068,600
Schedule Extended and BCWS revised due to ECP #1 in month 4 and ECP #2 in month 19.

Build #1, Integration Test

Project Month	Total BCWS	Schedule length in months	Total requirements	Software Requirements Completed to date	BCWS	BCWP	ACWP	CPI	SPI	Start Month	End Month
30	$10,640,000	5	612	51	$4,256,000	$933,976	$1,336,000	0.699	0.219	29	33
33	$10,640,000	5	612	350	$10,640,000	$6,409,639	$7,498,000	0.855	0.602	29	33
36	$10,640,000	8	612	581	$10,640,000	$10,640,000	$12,473,000	0.853	1.000	29	36

Original Schedule: Start Month 27, End Month 31, BCWS = $10,640,000
Final Schedule: Start Month 29, End Month 36, BCWS $10,640,000
Schedule Extended due to ECP #1 in month 4 and ECP #2 in month 19.

Build #1, Summary

Project Month	Total BCWS	BCWS	BCWP	ACWP	CPI	SPI
2	$2,128,000	$851,200	$893,760	$872,000	1.025	1.050
4	$17,920,800	$2,948,800	$3,073,908	$3,148,000	0.976	1.042
6	$17,920,800	$6,080,000	$6,115,954	$6,534,616	0.936	1.006
11	$17,920,800	$12,312,000	$10,683,282	$13,525,642	0.790	0.868
14	$33,880,800	$19,881,600	$18,602,179	$23,562,484	0.789	0.936
16	$33,880,800	$24,304,800	$24,359,834	$30,254,290	0.805	1.002
19	$35,241,200	$28,135,200	$28,109,271	$34,054,302	0.825	0.999
21	$35,241,200	$30,719,200	$30,712,991	$36,520,624	0.841	1.000
25	$50,137,200	$43,753,200	$42,432,372	$49,720,936	0.853	0.970
28	$50,137,200	$50,137,200	$47,825,752	$56,232,936	0.850	0.954
30	$60,777,200	$54,393,200	$51,071,176	$60,346,936	0.846	0.939
33	$60,777,200	$60,777,200	$56,546,839	$66,508,936	0.850	0.930
36	$60,777,200	$60,777,200	$60,777,200	$71,483,936	0.850	1.000

Original Schedule: Start Month 0, End Month 31, BCWS = $57,304,000
Final Schedule: Start Month 29, End Month 36, BCWS $60,777,200
Schedule Extended & BCWP revised due to ECP #1 in month 4 and ECP #2 in month 19.

Build #2, CSCI A, SW Requirements Analysis

Project Month	Total BCWS	Schedule length in months	Total requirements	System Requirements Completed to date	BCWS	BCWP	ACWP	CPI	SPI	Start Month	End Month
27	$1,216,000	4	48	33	$912,000	$836,000	$905,000	0.924	0.917	25	28
28	$1,216,000	4	48	45	$1,216,000	$1,140,000	$1,199,000	0.951	0.938	25	28
29	$1,216,000	4	48	48	$1,216,000	$1,216,000	$1,280,000	0.950	1.000	25	28

Original Schedule: Start Month 24, End Month 27, BCWS = $1,033,600
Final Schedule: Start Month 25, End Month 28, BCWS = $1,216,000
Schedule Extended & BCWP revised due to ECP #1 in month 4, ECP #2 in month 19 and ECP #3 in month 25.

Build #2, CSCI A, SW Design

Project Month	Total BCWS	Schedule length in months	Total requirements	Software Requirements Completed to date	BCWS	BCWP	ACWP	CPI	SPI	Start Month	End Month
27	$7,904,000	10	264	28	$790,400	$838,303	$795,000	1.054	1.061	27	36
28	$7,904,000	10	264	55	$1,580,800	$1,646,667	$1,590,000	1.036	1.042	27	36
29	$7,904,000	10	270	85	$2,371,200	$2,488,296	$2,411,000	1.032	1.049	27	36
33	$7,904,000	10	300	188	$5,532,800	$4,953,173	$5,809,000	0.853	0.895	27	36
37	$7,904,000	10	300	300	$7,904,000	$7,904,000	$9,128,000	0.866	1.000	27	36

Original Schedule: Start Month 26, End Month 34, BCWS = $6,688,000
Final Schedule: Start Month 27, End Month 36, BCWS = $7,904,000
Schedule Extended & BCWP revised due to ECP #1 in month 4, ECP #2 in month 19 and ECP #3 in month 25.

Build #2, CSCI A, C&UT

Project Month	Total BCWS	Schedule length in months	Total requirements	Software Requirements Completed to date	BCWS	BCWP	ACWP	CPI	SPI	Start Month	End Month
37	$9,179,711	9	313	111	$4,079,872	$3,255,425	$3,046,950	1.068	0.798	34	42
40	$9,179,711	9	313	203	$7,139,775	$5,953,614	$5,872,918	1.014	0.834	34	42
42	$10,933,591	11	333	263	$9,036,683	$7,713,303	$8,232,158	0.937	0.854	34	44
43	$10,933,591	11	333	297	$9,985,137	$9,751,581	$9,695,000	1.006	0.977	34	44
44	$10,933,591	11	333	331	$10,933,591	$10,867,924	$10,345,079	1.051	0.994	34	44
45	$10,933,591	11	333	333	$10,933,591	$10,933,591	$10,399,980	1.051	1.000	34	44

Original Schedule: Start Month 34, End Month 42, BCWS = $9,179,000
Final Schedule: Start Month 34, End Month 44, BCWS = $10,933,591
Schedule Extended & BCWP revised due to ECP #1 in month 4, ECP #2 in month 19, ECP #3 in month 25 and ECP #4 in month 40.

Build #2, CSCI A, CSCI Integration Test

Project Month	Total BCWS	Schedule length in months	Total requirements	Software Requirements Completed to date	BCWS	BCWP	ACWP	CPI	SPI	Start Month	End Month
43	$8,937,600	6	333	56	$1,489,600	$1,503,020	$1,500,000	1.002	1.009	43	48
44	$8,937,600	6	333	112	$2,979,200	$3,006,040	$3,010,000	0.999	1.009	43	48
45	$8,937,600	6	333	168	$4,468,800	$4,509,059	$4,500,000	1.002	1.009	43	48
48	$8,937,600	6	333	333	$8,937,600	$8,937,600	$8,937,600	1.000	1.000	43	48

Original Schedule: Start Month 43, End Month 48, BCWS = $8,937,600
Final Schedule: Start Month 43, End Month 48, BCWS = $8,937,6001

Build #2, CSCI A Summary

Project Month	Total BCWS	BCWS	BCWP	ACWP	CPI	SPI
27	$9,120,000	$1,702,400	$1,674,303	$1,700,000	0.985	0.983
28	$9,120,000	$2,796,800	$2,786,667	$2,789,000	0.999	0.996
29	$9,120,000	$3,587,200	$3,704,296	$3,691,000	1.004	1.033
33	$9,120,000	$6,748,800	$6,169,173	$7,089,000	0.870	0.914
37	$18,299,711	$13,199,872	$12,375,425	$13,454,950	0.920	0.938
40	$18,299,711	$16,259,775	$15,073,614	$16,280,918	0.926	0.927
42	$20,053,591	$18,156,683	$16,833,303	$18,640,158	0.903	0.927
43	$28,991,191	$20,594,737	$20,374,601	$21,603,000	0.943	0.989
44	$28,991,191	$23,032,791	$22,993,963	$23,763,079	0.968	0.998
45	$28,991,191	$24,522,391	$24,562,650	$25,307,980	0.971	1.002
48	$28,991,191	$28,991,191	$28,991,191	$29,745,580	0.975	1.000

Original Schedule: Start Month 24, End Month 48, BCWS = $25,838,200
Final Schedule: Start Month 25, End Month 48, BCWS = $28,991,191
Schedule Extended & BCWP revised due to ECP #1 in month 4, ECP #2 in month 19, ECP #3 in month 25 and ECP #4 in month 40.

Build #2, CSCI B Summary

Project Month	Total BCWS	BCWS	BCWP	ACWP	CPI	SPI
27	$9,120,000	$1,702,400	$1,674,303	$1,700,000	0.985	0.983
28	$9,120,000	$2,796,800	$2,786,667	$2,789,000	0.999	0.996
29	$9,120,000	$3,587,200	$3,704,296	$3,691,000	1.004	1.033
33	$9,120,000	$6,748,800	$6,169,173	$7,089,000	0.870	0.914
37	$18,299,711	$13,199,872	$12,375,425	$13,454,950	0.920	0.938
40	$18,299,711	$16,259,775	$15,073,614	$16,280,918	0.926	0.927
42	$20,053,591	$18,156,683	$16,833,303	$18,640,158	0.903	0.927
43	$28,991,191	$20,594,737	$20,374,601	$21,603,000	0.943	0.989
44	$28,991,191	$23,032,791	$22,993,963	$23,763,079	0.968	0.998
45	$28,991,191	$24,522,391	$24,562,650	$25,307,980	0.971	1.002
48	$28,991,191	$28,991,191	$28,991,191	$29,745,580	0.975	1.000

Original Schedule: Start Month 24, End Month 48, BCWS = $25,838,200
Final Schedule: Start Month 25, End Month 48, BCWS = $28,991,191
Schedule Extended & BCWP revised due to ECP #1 in month 4, ECP #2 in month 19, ECP #3 in month 25 and ECP #4 in month 40.

Build #2, Integration Test

Project Month	Total BCWS	Schedule length in months	Total requirements	Software Requirements Completed to date	BCWS	BCWP	ACWP	CPI	SPI	Start Month	End Month
49	$12,157,765	6	666	74	$2,026,294	$1,350,863	$1,950,000	0.693	0.667	49	54
51	$12,157,765	6	666	222	$6,078,882	$4,052,588	$5,800,000	0.699	0.667	49	54
54	$12,157,765	6	666	444	$12,157,765	$8,105,176	$11,750,000	0.690	0.667	49	54
57	$12,157,765	6	666	666	$12,157,765	$12,157,765	$17,635,000	0.689	1.000	49	54

Original Schedule: Start Month 49, End Month 54, BCWS = $12,157,765
Final Schedule: Start Month 49, End Month 57, BCWS = $12,157,765

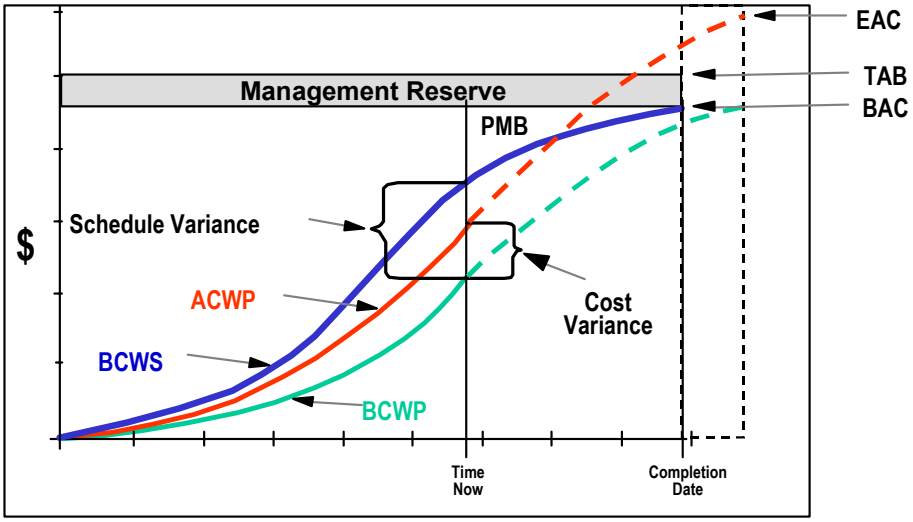
Build #2, Summary

Project Month	Total BCWS	BCWS	BCWP	ACWP	CPI	SPI
27	$18,240,000	$3,404,800	$3,348,606	$3,400,000	0.985	0.983
28	$18,240,000	$5,593,600	$5,573,333	$5,578,000	0.999	0.996
29	$18,240,000	$7,174,400	$7,408,593	$7,382,000	1.004	1.033
33	$18,240,000	$13,497,600	$12,338,347	$14,178,000	0.870	0.914
37	$36,599,422	$26,399,743	$24,750,849	$26,909,900	0.920	0.938
40	$36,599,422	$32,519,550	$30,147,229	$32,561,836	0.926	0.927
42	$40,107,182	$36,313,366	$33,666,607	$37,280,316	0.903	0.927
43	$57,982,382	$41,189,474	$40,749,202	$43,206,000	0.943	0.989
44	$57,982,382	$46,065,582	$45,987,927	$47,526,158	0.968	0.998
45	$57,982,382	$49,044,782	$49,125,301	$50,615,960	0.971	1.002
48	$57,982,382	$57,982,382	$57,982,382	$59,491,160	0.975	1.000
49	$70,140,147	$60,008,676	$59,333,245	$61,441,160	0.966	0.989
51	$70,140,147	$64,061,264	$62,034,970	$65,291,160	0.950	0.968
54	$70,140,147	$70,140,147	$66,087,558	$71,241,160	0.928	0.942
57	$70,140,147	$70,140,147	$70,140,147	$77,126,160	0.909	1.000

Original Schedule: Start Month 25, End Month 54, BCWS = $63,834165
Final Schedule: Start Month 25, End Month 57, BCWS = $70,140,147
Schedule Extended & BCWP revised due to ECP #1 in month 4, ECP #2 in month 19, ECP #3 in month 25 and ECP #4 in month 40.

APPENDIX K. EARNED VALUE MANAGEMENT GOLD CARD

Defense Systems Management College
Earned Value Management Gold Card



VARIANCES *(Favorable is positive, Unfavorable is negative)*

- Cost Variance — CV = BCWP – ACWP — CV % = CV / BCWP
- Schedule Variance — SV = BCWP – BCWS — SV % = SV / BCWS
- Variance at Completion — VAC = BAC – EAC

PERFORMANCE INDICES *(Favorable is > 1.0, Unfavorable is < 1.0)*

- Cost Efficiency — CPI = BCWP / ACWP
- Schedule Efficiency — SPI = BCWP / BCWS

OVERALL STATUS

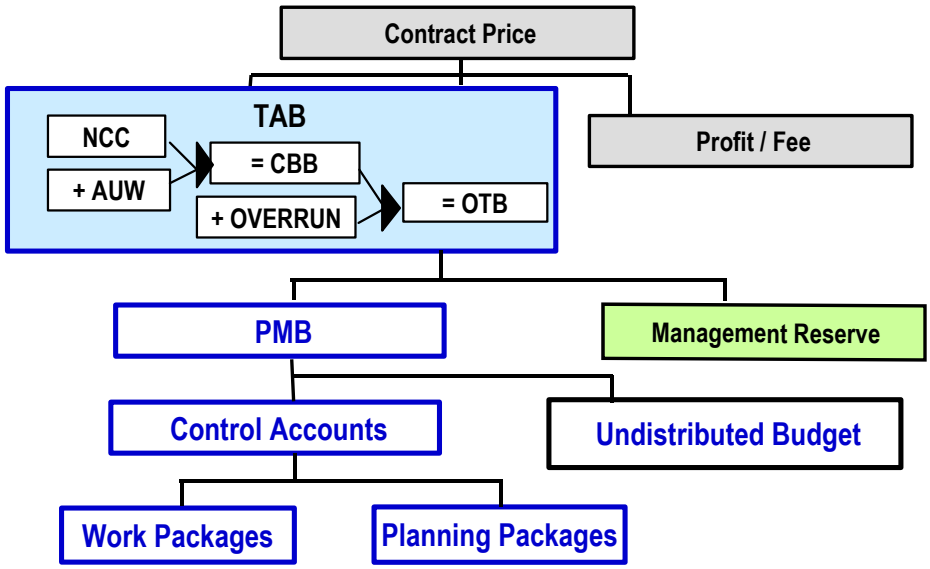
- Percent Complete = $\frac{\text{BCWP CUM}}{\text{BAC}}$
- Percent Spent = $\frac{\text{ACWP CUM}}{\text{BAC}}$

TO COMPLETE PERFORMANCE INDEX (TCPI)

$$\text{TCPI}_{(\text{EAC})} = \frac{\text{WORK REMAINING}}{\text{COST REMAINING}} = \frac{\text{BAC – BCWP CUM}}{\text{EAC – ACWP CUM}}$$

ESTIMATE AT COMPLETION (EAC = ACWP + Estimate for Remaining Work)

$$\text{EAC}_{\text{CPI}} = \frac{\text{BAC}}{\text{CPI CUM}} \qquad \bullet\ \text{EAC}_{\text{Composite}} = \text{ACWP CUM} + \frac{\text{BAC – BCWP CUM}}{(\text{CPI CUM}) \bullet (\text{SPI CUM})}$$



TERMINOLOGY

NCC – Negotiated Contract Cost	Contract price less profit / fee
AUW – Authorized Unpriced Work	Work authorized to start, not yet negotiated
CBB – Contract Budget Base	Sum of NCC and AUW
OTB – Over Target Baseline	Sum of CBB and recognized overrun
TAB – Total Allocated Budget	Sum of all contract budgets - NCC,CBB or 0TB (includes MR)
BAC – Budget At Completion	Cumulative BCWS - total end point of PMB (excludes MR)
PMB – Performance Measurement Baseline	Contract time-phased, budgeted work plan (excludes MR)
MR – Management Reserve	Contractor PM's Contingency budget
UB – Undistributed Budget	Broadly defined activities not yet distributed to CAs
CA – Control Account	Contractor key management control point - CWBS element
WP – Work Package	Near-term, detail-planned activities within a CA
PP – Planning Package	Far-term CA activities not yet defined into detail Work Packages
BCWS – Budgeted Cost for Work Scheduled	Value of work scheduled -- PLAN
BCWP – Budgeted Cost for Work Performed	Value of work completed -- EARNED VALUE
ACWP – Actual Cost of Work Performed	Cost of work completed -- ACTUAL COSTS INCURRED
EAC – Estimate At Completion	Estimate of total contract costs

EVM POLICY (DOD 5000.2-R)

ALTERNATIVE EV MANAGEMENT APPLICATIONS

LEVEL 1. EVMS Industry Standards Management Application

Contractor management system certified as meeting Industry Standards

- Required for non-FFP contract exceeding $73M RDT&E or $315M in procurement (CY00$).
- PM may apply to contracts below-threshold —consider benefits, risk and criticality.
- Contractor must establish, maintain, and use a system that meets the the 32 Industry Standards.
- Cost Performance Report (CPR) delivered as a CDRL item.
  - 5 Formats (WBS, Organization, Baseline, Staffing, and Explanations)

LEVEL 2. C/SSR Management Application

Contractor Management system not certified

- Required for non-FFP contract exceeding $6.3M (CY00$) and 12 months in length.
- 'Reasonably objective' EV methods acceptable, traceability at higher level (CA vs WP)
- The CPR or the Cost/Schedule Status Report (C/SSR) delivered as a CDRL item.

EVM Home Page — http://www.acq.osd.mil/pm/
DSMC EV E-Mail Address — EVM@DSMC.DSM.MIL
DSMC EV Phone No. — (703) 805–2848/2968 (DSN 655)

June 2000

Intentionally left blank