SYSTEM ENGINEERING THE MISSION CRITICAL

SOFTWARE RELEASE DECISION

Approved by:

_____

Dr. Mitch Thornton, Ph.D. – Committee Chairman

_____

Dr. Jerrell Stracener, Ph.D. – Dissertation Director

_____

Dr. Khaled Abdelghany, Ph.D.

_____

Dr. W. D. (Dave) Bell, D. Eng.

_____

Dr. George Chollar, Ph.D.

_____

Dr. Stephen Szygenda, Ph.D.

SYSTEM ENGINEERING THE MISSION CRITICAL SOFTWARE RELEASE DECISION

A Dissertation Presented to the Graduate Faculty of

The School Of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Applied Science

by

Tim Woods

(B.S.E.E., Michigan Technological University)
(M.S.E.M., Southern Methodist University)
(M.S.S.E., Southern Methodist University)

December 18, 2010

Woods, Tim B.S., Michigan Technological University, 1987
M.S., Southern Methodist University, 1995
M.S., Southern Methodist University, 2005

SYSTEM ENGINEERING THE MISSION CRITICAL
SOFTWARE RELEASE DECISION

Advisor: Dr. Jerrell T. Stracener

Doctor of Philosophy conferred December 18, 2010

Dissertation completed October 15, 20010

Mission critical software can be defined as software that a system requires to perform its mission. Mission critical software may also be safety critical software. Whether mission or safety critical, software is an integrated, crucial aspect in the development of today's complex systems. If the software fails, or is not able to perform its intended purpose, there is a risk of system failure. Critical software requires additional rigor during the design, development, release, and test life cycles to help prevent system failures. This additional rigor may be imparted through proven processes and analytical methodologies.

Systems engineering principles are used to develop an analysis driven, quantifiable, Mission Critical Release Readiness Methodology (MCRRM) for releasing mission critical software, focusing on the decision to release software from the development process to the customer. A software release occurs in order to document the current software and transfer the software to the customer. Multiple factors affect when to release the software. The mission critical release readiness methodology is developed

through rigorous application of the systems engineering process starting from the customer needs through design, development, implementation, test, and verification and validation of the process itself. The application of the methodology is intended for projects with multiple subsystems with varying levels of critical software being released as part of an overall system. Validation and Verification (V&V) of the methodology was accomplished via thorough testing and representative case studies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AHP – Analytic Hierarchy Process

ASD – Adaptive Software Development

ASRD – Analyze Software Release Decision

BIOS – Basic Input/ Output System

BOGSAT – Bunch Of Guys/Gals Sitting Around a Table

COCOMO – COnstructive COst Model

CMMI – Capability Maturity Model Integration

DoD – Department of Defense

DSDM – Dynamic Systems Development Method

FFBD – Functional Flow Block Diagram

GNU – GNU's Not Unix

INCOSE – International Council On Systems Engineering

IO – Input/Output

KSLOC – Thousands of Source Lines Of Code

LOC – Lines Of Code

MCRRM – Mission Critical Release Readiness Methodology

NASA – National Aeronautics and Space Administration

N/A – Not Applicable

OS – Operating System

PERT – Program Evaluation and Review Technique

QFD – Quality Function Deployment

SE – Systems Engineering

SEER-SEM – System Evaluation and Estimation of Resources – Software Estimating Model

SEP – Systems Engineering Process

SLOCS – Source Lines Of Code

SoS – Systems of Systems

SRGM – Software Reliability Growth Model

SRM – Software Release Manager

SME – Subject Matter Experts

SW – Software

TOPSIS – Technique for Order Preference by Similarity to Ideal Solution

TPM – Technical Performance Measurement

UI – User Interface

V&V – Verification and Validation

VCRM – Verification Cross Reference Matrix

ACKNOWLEDGEMENTS

This journey was not taken by me alone. I was accompanied by family, committee members, friends, and co-workers during this journey. No one was more a part of this journey or more important than my wife, Dawn Woods. She did not journey as a passenger, but as my co-pilot, navigator, and occasionally even as pilot, helping to guide and direct me when my path strayed and taking control when other distractions made the journey seem impassable. Her guidance, support, and love gave me strength, and for that, I am grateful.

My parents, Tom and Karla Woods, provided the foundation of the journey and faith along the journey. The Woods and Tormohlen families, especially Ed and Eleanor Tormohlen, inspired me and provided encouragement when needed.

My committee guided and mentored me on this journey and I could not have made it without them. I thank them for all they did. Dr. Stracener not only provided thesis advising, but also guidance and mentoring that will be useful in the years to come. Dr. Thornton provided inspiration for reaching far beyond what I thought was possible. Dr. Szygenda's guidance and viewpoint made the journey bearable. Dr. Abdelghany provided critical guidance during the journey and especially as the journey was nearing completion. Dr. Bell provided critical direction in my decision to begin the journey and welcomed consultation during the journey. Dr. Chollar's insight and analytical guidance kept the journey on track

and within reach. I am humbled and very grateful for all my committee did on my behalf and again, thank them.

I cannot thank all my friends and co-workers enough for their inquiries on my status and continued support during the journey. I would like to especially thank Dr. Skinner and Dr. Zummo for their assistance and friendship during our educational journeys. They allowed me glimpses into their journey and provided much needed consultation during mine.

My daughters, Cayla and Corinna, were passengers and inspirational on this part of my educational journey. I hope they see education as a journey to take and not a merely a task and hope that I can provide inspiration to them during their personal journeys, as they have inspired me.

Chapter 1

INTRODUCTION

Mission critical software can be defined as software that a system requires to perform its mission. Mission critical software may also be safety critical software. Whether mission or safety critical, software is an integrated, crucial aspect in the development of today's complex systems. If the software fails, or is not able to perform its intended purpose, there is a risk of system failure. Critical software requires additional rigor during the design, development, release, and test life cycles to help prevent system failures. This additional rigor may be imparted through proven processes and analytical methodologies.

Systems engineering principles are used to develop an analysis driven, quantifiable, Mission Critical Release Readiness Methodology (MCRRM) for releasing mission critical software, focusing on the decision to release software from the development process to the customer. A software release occurs in order to document the current software and transfer the software to the customer. Multiple factors affect when to release the software. The mission critical release readiness methodology is developed through rigorous application of the systems engineering process starting from the customer needs through design, development, implementation, test, and verification and validation of the process itself. The application of the methodology is intended for projects with multiple subsystems with varying levels of critical software being released

as part of an overall system. Validation and Verification (V&V) of the methodology was accomplished via thorough testing and representative case studies.

A software release consumes both time (schedule and personnel time) and resources (personnel, computers, media, etc.). Software releases can also negatively or positively effect customer satisfaction. The developed mission critical release readiness methodology analyzes the software release decision as a system in itself and uses analytical methodologies to determine the optimal system (software release decision) solution based on the current program and system states, while providing the software release decision maker with quantitative analysis on the release decision with the intent of reducing overall program cost, schedule, and risk, while increasing customer satisfaction.

1.1 Background

1.1.1 Software

Merriam-Webster's online dictionary defines software as something used or associated with and usually contrasted with hardware: as the entire set of programs, procedures, and related documentation associated with a system and especially a computer system; specifically: computer programs [14]. In today's technological society, software is ubiquitous. Software is used in everyday items from cars, to microwaves, to pacemakers. Software runs on some type of hardware. The type of

hardware, microcontroller, personal computer, etc., being used does not concern us at this time. What may concern us is what type of software is being developed and released, with type referring to one of the following: 1) Basic Input/ Output System (BIOS); 2) Operating System (OS); or 3) application software.

The BIOS normally comes from the hardware vendor and is provided as a part of the hardware to allow the other types of software to access the hardware's specific capabilities. BIOS' major task is to load the operating system for the hardware.[26] At some time in the development phase, computer hardware does not have a operating system loaded. It is the BIOS that permits the hardware to load operating system software.

BIOS and the operating system software can be further classified as systems software as they both control the hardware functions [27]. The operating system manages the resources of the hardware – both hardware and software while allowing the application software to access those same resources in a stable and consistent environment [27]. Widely known operating systems include Microsoft Windows®, UNIX, Linux, and Mac OS.

Application software is different than system software as it is written for the user to perform some task or set of tasks using the operating system to manage the hardware resources [27]. Application software does not normally control the hardware resources directly. Examples of application software uses are: listen to and record music; develop software; develop a presentation; write a document; surf the web; manage data; etc.

The interactions of BIOS, operating system and application software are shown in Figure 1 – BIOS, OS, and Application Software. BIOS software is shown interacting

with portions of the hardware- just enough to get the operating system loaded. The operating system interacts with all of the hardware. And the application software, shown with some example applications, is a byproduct of the interactions of the BIOS and operating system software.

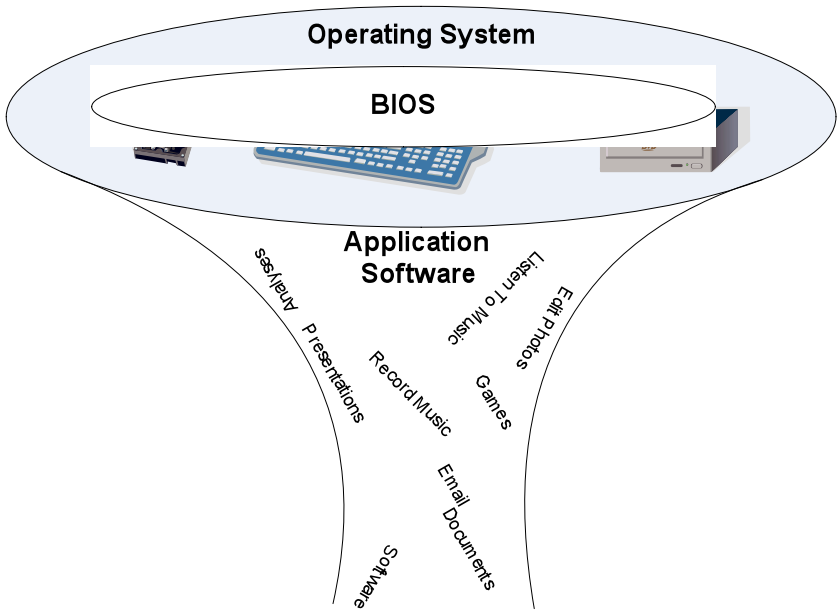

Figure 1 – BIOS, OS, and Application Software

The BIOS software used by one vendor does not necessarily have to match that of another vendor, but the software will be similar. Other aspects of BIOS are that it is very specialized software performing well known actions and therefore does not require many updates.

Operating system software continues to evolve and change – some changes are to add features and capabilities, some changes are due to security exploits of the software. The operating system interacts with both the BIOS and application software, but the

operating system software is written for a particular computer or application and is not normally encompassed of multiple releases of differing software. So while the operating system appears to be more complex than BIOS software, it would appear there is a more complex release problem waiting to be solved.

Application software interacts with the user, which does add complexity to the software just due to human interaction. Adding to the complexity is interacting with OS software. The problem of when to release application software could be complex. Just to insure the problem is not overly simple, what if application software interacted with the user, OS and other application software?

Imagine a system consisting of multiple other smaller systems and several of those systems are composed of hardware and application software. Assume this imagined system's smaller systems are interrelated and require some aspect of the other systems to perform their intended purpose. Now imagine having to decide when to release the system, which of course means deciding when to release the smaller systems software. Now the application software is dealing with the OS, the user, and other application software. Now there is a problem to solve.

Before getting too comfortable with this problem, let's add to the complexity by releasing mission critical software. Depending upon the application and the system using the software, mission critical software may also be flight critical or even safety critical. Now we have a problem to solve – how to decide when to release mission critical software for a system composed of smaller systems.

1.1.2 Systems Engineering Defined

The International Council On Systems Engineering (INCOSE) defines systems engineering as an interdisciplinary approach and means to enable the realization of successful systems [15]. The realization of successful systems requires systems engineer to be able to focus on development of hardware and software, system integration, system test, system support and reliability, and system disposal. INCOSE further states that systems engineering focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem: Operations; Cost & Schedule; Performance; Training & Support; Test; Manufacturing; and Disposal.

Engineering has been practiced for years – traceable back to stone tools and man's desire for fire [32], but systems engineering was not formalized until more recently. Systems Engineering is believed to have roots born in the 1940s in the telecommunication industry [18] and [31] and in the military development process used on the ballistic missile programs of the mid-1950s [16]. With its basis in processes and formal methods, systems engineering's formalization in the telecommunications and ballistic missile systems was a natural extension of engineering principles. INCOSE, a systems engineering professional organization, was started in 1990 [33] and as of December 2008 had over 6,700 members [35].

Although systems engineering is known for its processes and procedures, it is not a one size fits all with respect to processes and methodologies. Systems engineers have developed several processes to apply, depending upon the system under development.

Additionally, systems engineering allows for customization of processes to align with the system and problem being worked.

As part of this research, systems engineering processes were researched and reviewed for their applicability to deciding when to release software. A systems engineering process was then chosen and a customization of the process developed and applied to the problem under study.

Figure 2 illustrates the fundamental systems engineering process and is described below from systems engineering fundamentals [17] and [16].

> The Systems Engineering Process (SEP) is a comprehensive, iterative and recursive problem solving process, applied sequentially top-down by integrated teams. It transforms needs and requirements into a set of system product and process descriptions, generate information for decision makers, and provides input for the next level of development. The process is applied sequentially, one level at a time, adding additional detail definition with each level of development.

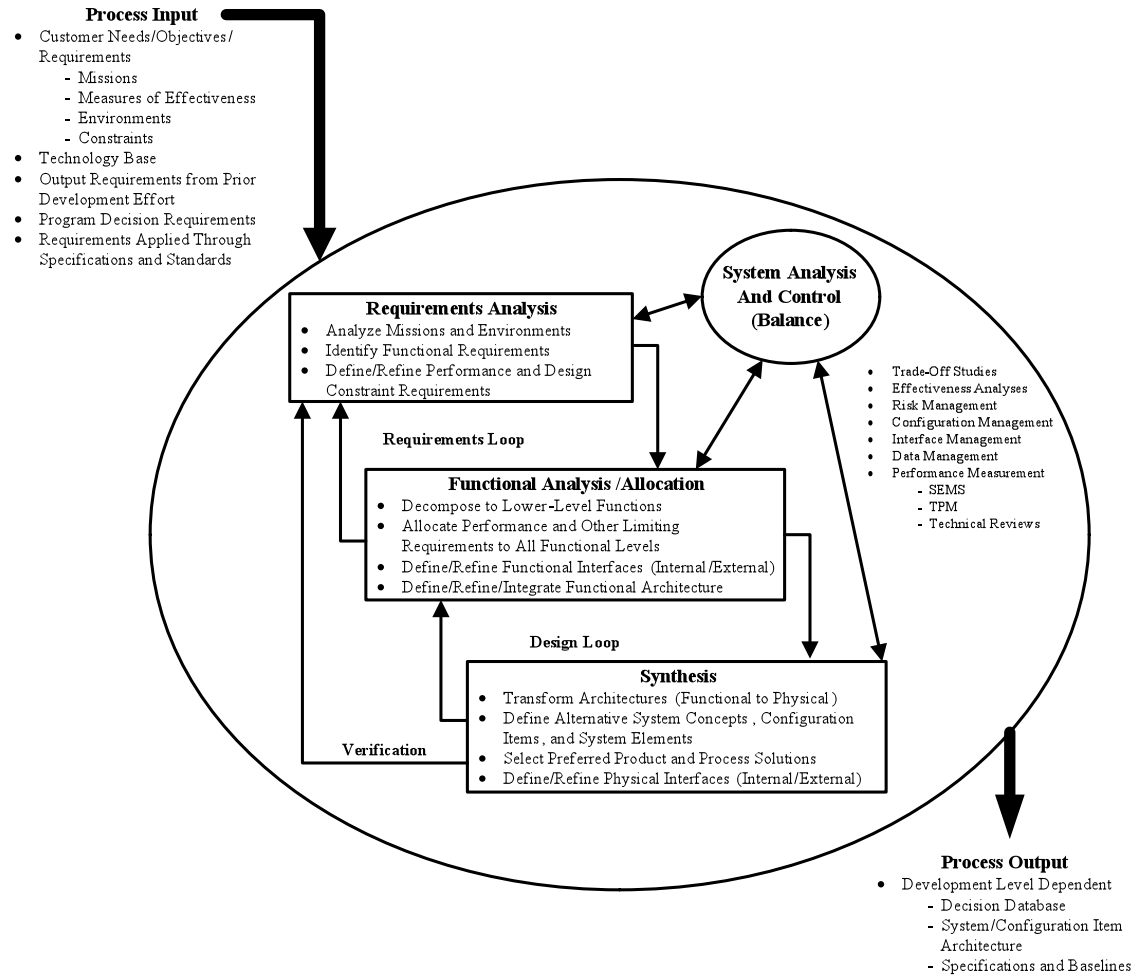

Figure 2 – The Fundamental Systems Engineering Process

Another systems engineering process is the SIMILAR Process, shown in Figure 3 – SIMILAR Process (Bahil and Gissing). SIMILAR is the acronym for the seven steps of the process: **S**tate the Problem, **I**nvestigate Alternatives, **M**odel the System, **I**ntegrate, **L**aunch the System, **A**ssess the Performance, and **R**e-evaluate.

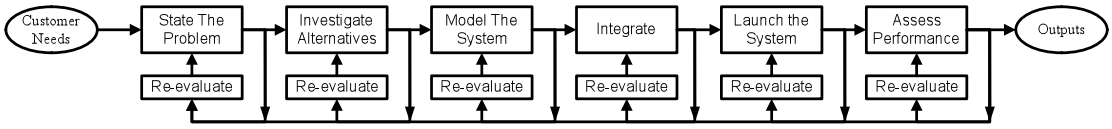

Figure 3 – SIMILAR Process (Bahil and Gissing)

A modified Forsberg and Mooz systems engineering "Vee Model" is shown in Figure 4 [91]. The "Vee Model" is a top down, bottom up approach useful in decomposing systems into finer levels of details and then integrating and testing at those levels on the way up the left side of the "Vee".

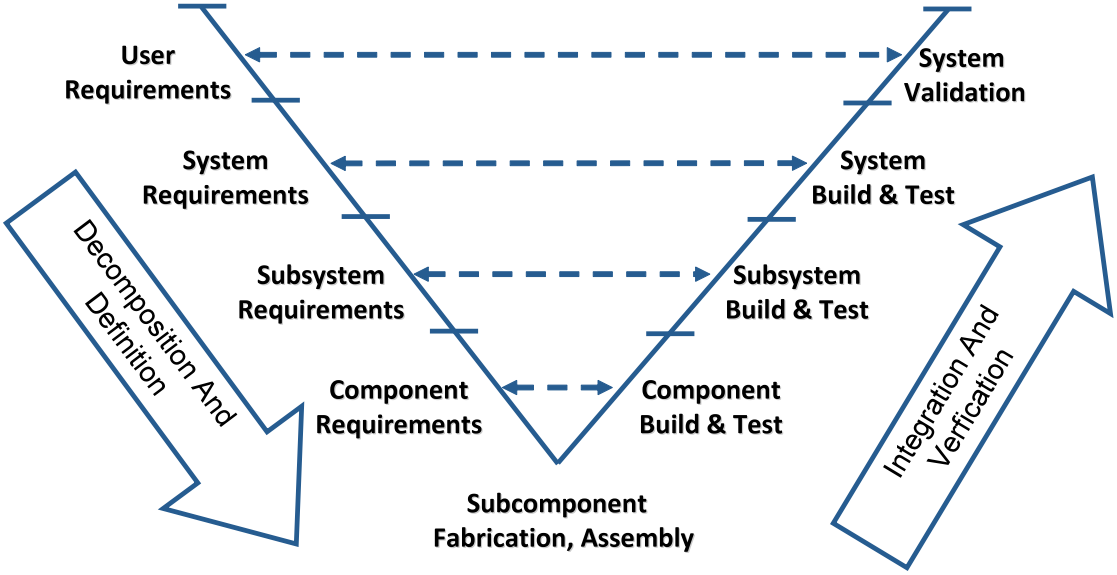

Figure 4 – Systems Engineering "Vee" Model

Several processes have been developed for applying systems engineering to software development. Boehm's Spiral development model is one such model and is

shown in Figure 5. The Spiral model is useful in projects where the design and therefore the requirements are not fully developed at program onset [36].

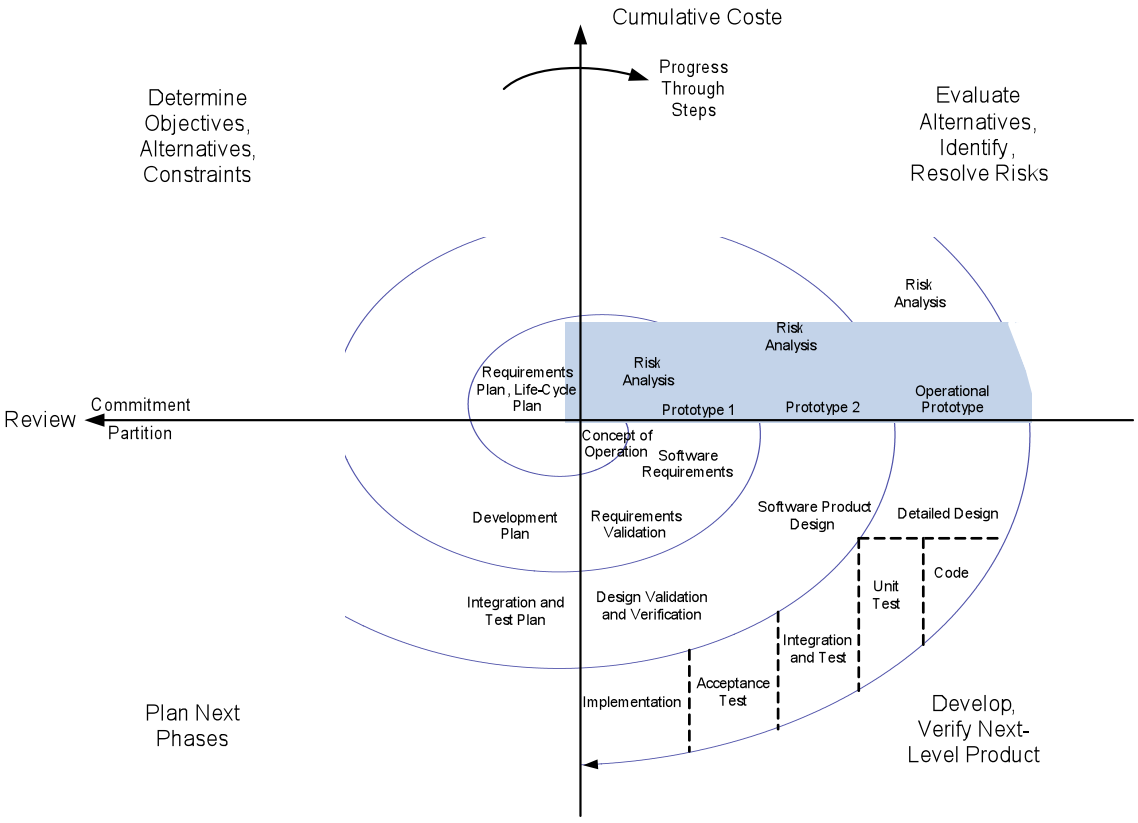

Figure 5 – Spiral Model

The spiral model was developed specifically for software development. Although the problem under consideration, assisting with the software release decision, deals with software, it is the process and decision making regarding the problem that is focus of this paper. Any of the system engineering models and processes could be adapted to the how to decide when to release problem, but the fundamental systems engineering process

aligns well with implementing a decision process and therefore will be used as the base model for the problem of how to decide when to release software.

1.1.3 Software And Systems Engineering

The added complexity of the software in systems today adds to the costs of developing a system, but not adhering to strong software development processes and not following good requirements management processes will increase the development costs also [20]. With complex, mission critical software complicating release planning and decision making, systems engineering, and its focus on realizing successful systems, is used to develop a methodology for assisting in the decision making process for releasing mission critical software.

The software and systems engineering fields are not currently well integrated, although there are several initiatives underway to do just that, e.g. Capability Maturity Model Integration (CMMI). And we are not trying to integrate the fields, just applying systems engineering to solve a software process problem. Both software and systems engineering have their own domains and tools, processes and best practices unique to the tasking in their area. This paper borrows some tools and processes from each field and applies them to solve a problem.

Software has evolved since its start with Jaquard using punch cards in a weaving loom to perform predefined tasks in the early 1800's [22]. Interesting enough, 180 years after software's start; the American Society for Engineering Management was using computer cards for a mailing list [23] and students on college campuses across the United

States were still using punch cards in their programming classes. As Platt says in his book, "Fifteen years ago – even ten – ordinary people didn't use software in their daily lives…That's changed completely, almost overnight in societal terms and seemingly without our noticing it." [58] The tools used to develop software have evolved beyond punch cards, but this is not a discussion on software development or its difficulties. As the INCOSE Modeling and Systems engineering workgroup stated, "Software doesn't fly and wings don't go far on their own [24]." At some point, the software and hardware must be integrated to allow the complete system to operate. In order for this integration to occur, the software must make it through the development process and then through a release process to be released for use. The problem is the difficulties encountered when releasing mission critical software for its intended purpose. The problem area will be researched and a solution will be developed.

Releasing software sounds easy enough, just decide when to release and move on, but the release decision is multi-dimensional problem. If the software is released too soon, it may not be able to support its intended purpose, be too buggy or customers may not upgrade because they just bought the last version. Release the software too late and customers may be lost through competitors releasing earlier versions, customers may not upgrade because the older version no longer works, or the purpose for the software has been lost. Throw in contractual, fiscal, personnel, and resource issues and the release decision process difficulty increases.

Figure 6 shows the IEEE Std 610.12-1990 definition for a sample software life cycle [86]. A generic software process can be derived from the software life cycle. Combining Concept Exploration and the Requirements cycles into one process activity,

dividing the Installation and Checkout cycle into two process activities, and combining Operation and Maintenance and Retirement cycles, a generic software process can be derived. A top-level functional flow block diagram for the software development process is shown in Figure 7.

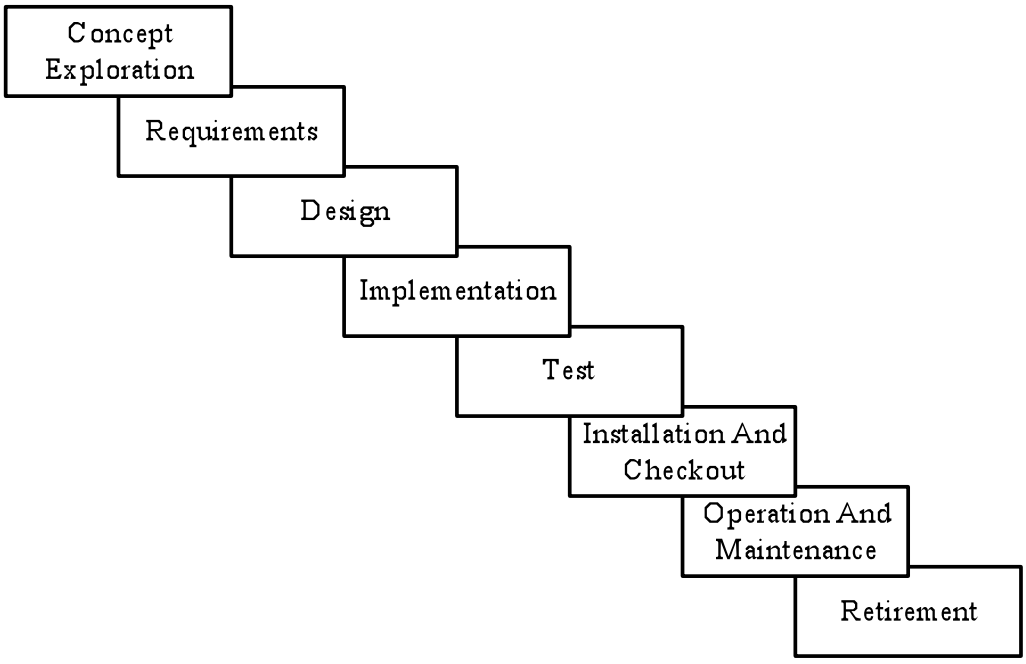

Figure 6 – IEEE Std 610.12-1990 Sample Software Life Cycle

A functional flow block diagram conveys what must happen in a time sequence for the item being diagramed [16]. Additionally, a functional flow block diagram may be used to assist in deciding the functions the item must accomplish and to provide a time sequence for the item. A functional flow block diagram also allows the user to develop the functions of the item under development and show the functions and sub-functions in a graphical representation. The functional flow block diagram will be used throughout

the development of the mission critical release readiness methodology. The derived software process shown consists of 7 functions: 1.0 – Develop Software Requirements; 2.0 – Design Software; 3.0 – Develop Software; 4.0 – Test Software; 5.0 Release Software; 6.0 – Deliver Software; and 7.0 – Support Software. Although all aspects of software development factor into the software release decision, the mission critical release readiness methodology is in regards to function 5.0 – Release Software. Other aspects of software development will be discussed as required to support consideration of how to decide when to release the software.

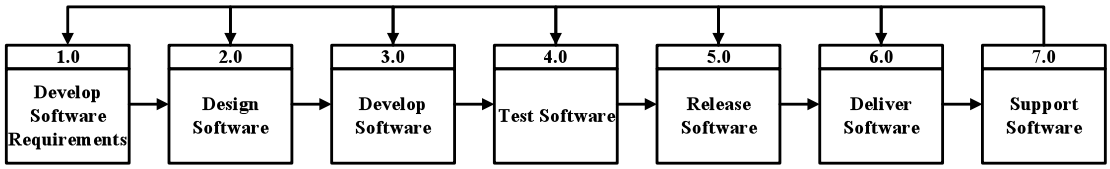

Figure 7 – Derived Software Development Process

The Release Software function consists of four sub-functions. The top-level software development process and the release software sub-function are shown in Figure 8. The four sub-functions of Release Software are: 5.1 – Analyze Software Release Decision; 5.2 – Decide to Release Software; 5.3 – Receive Approval To Release Software; 5.4 – Perform Software Release. The mission critical release readiness methodology develops processes and methods for the release software sub-function 5.1 – Analyze Software Release Decision. The remaining release software sub-functions are not developed further in this discussion, other than as required to support sub-function 5.1.

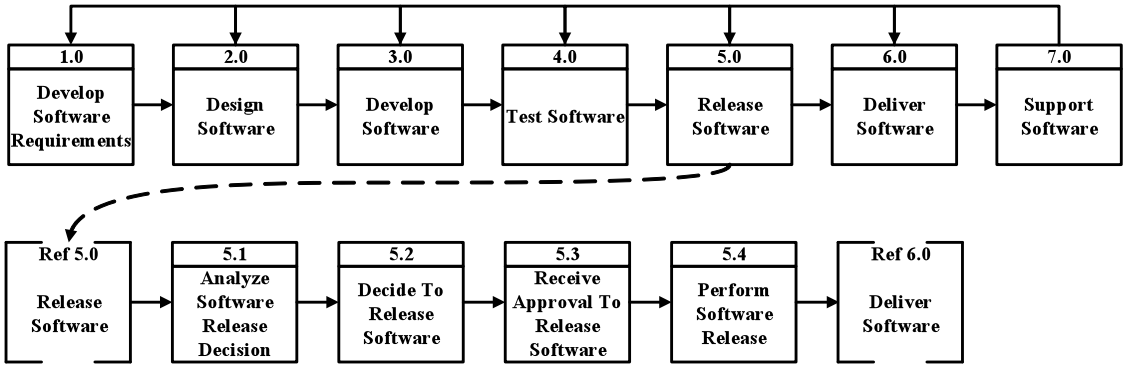

Figure 8 – Release Software Functional Block Diagram

The software release decision maker decides when to release the software. The decision maker may be a software manager, a project manager, or maybe even a program director. Again, software releasing sounds easy enough, just decide when to release and move on. Easy enough, that is, until one encounters change.

> Everything in software changes. The requirements change. The design changes. The business changes. The technology changes. The team changes. The team members change. The problem isn't change, because change is going to happen; the problem, rather, is our inability to cope with change.[25]

Changes during the software development process complicate the software release decision. Fortunately, the decision maker has a multitude of tools at their disposal to assist in the release decision process. Unfortunately for the decision maker, the ultimate decision still rests upon them. The software release decision maker must decide not only

the usefulness of any tool inputs, but the timing of the release in the perspective user's eyes, the effect on the organization of releasing software, the ability of the software to support its intended purpose, and how to overcome their own personal views on when the software should be released.

A systems engineering approach will be used in considering how to decide when to release software. The systems engineering approach aligns well with problem solving and will provide useful foundations from which to consider. Although there are many approaches to applying systems engineering, the fundamental systems engineering approach will be used to solve the software release system. The fundamental systems engineering process is tailored to fit how to decide when to release the software.

As a first step, the process used to identify, analyze, solve, and verify and validate the methodology was developed. The process develop is given in Figure 9.

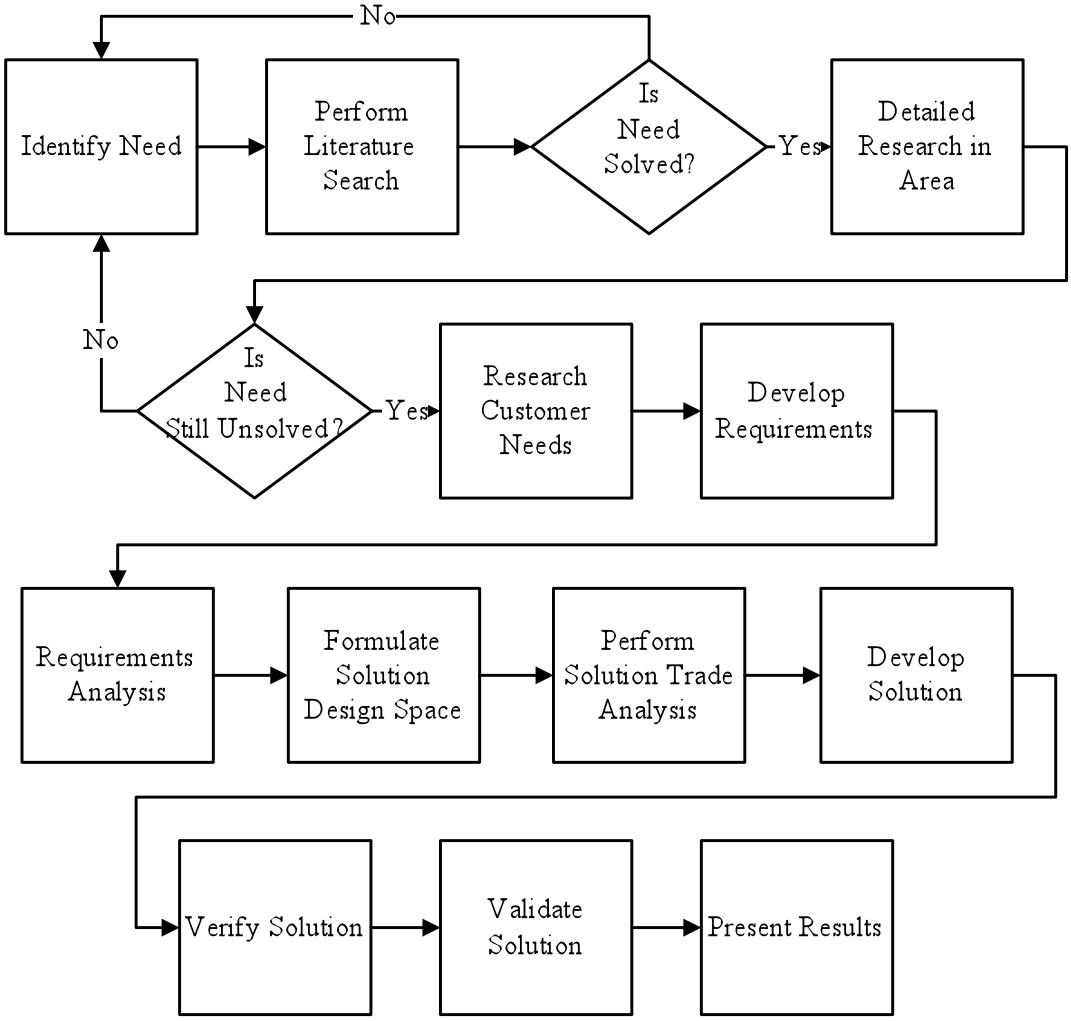

Figure 9 – Need Identification And Solution Response Process

The process developed defines a need identification step, with a control loop through a literature search to determine if the need identified has been solved. If the literature search did not find an adequate solution, detailed research on the need will be performed to determine if the need is still not adequately solved. If the detailed research determines the need is not adequately solved, customer needs are researched to determine the validity of the need and whether an adequate solution is developable. Next,

requirements are developed and analyzed based upon the customer needs researched previously. After requirements analysis, a design space of possible solutions is mapped to assist in determining alternative solutions and their relations to the other identified solutions. Given the design space, trades are performed to determine a solution that appears to fill the customer needs and developed requirements. The chosen solution's design is developed and tested for operability. The solution is then verified against the requirements and then validated against the customer needs. Finally, the results are presented.

The developed need identification and solution process is a tailoring of the fundamental systems engineering process. The fundamental systems engineering process consists of the following sub-processes: 1) *Process Inputs*; 2) *Requirements Analysis*; 3) *System Analysis and Control*; 4) *Functional Analysis/Allocation*; 5) *Synthesis*; 6) *Verification*; and 7) *Process Output*. The fundamental systems engineering process is shown in Figure 10. As a matter of convention, sub-processes will be indicated by italicized text for the remainder of this paper.

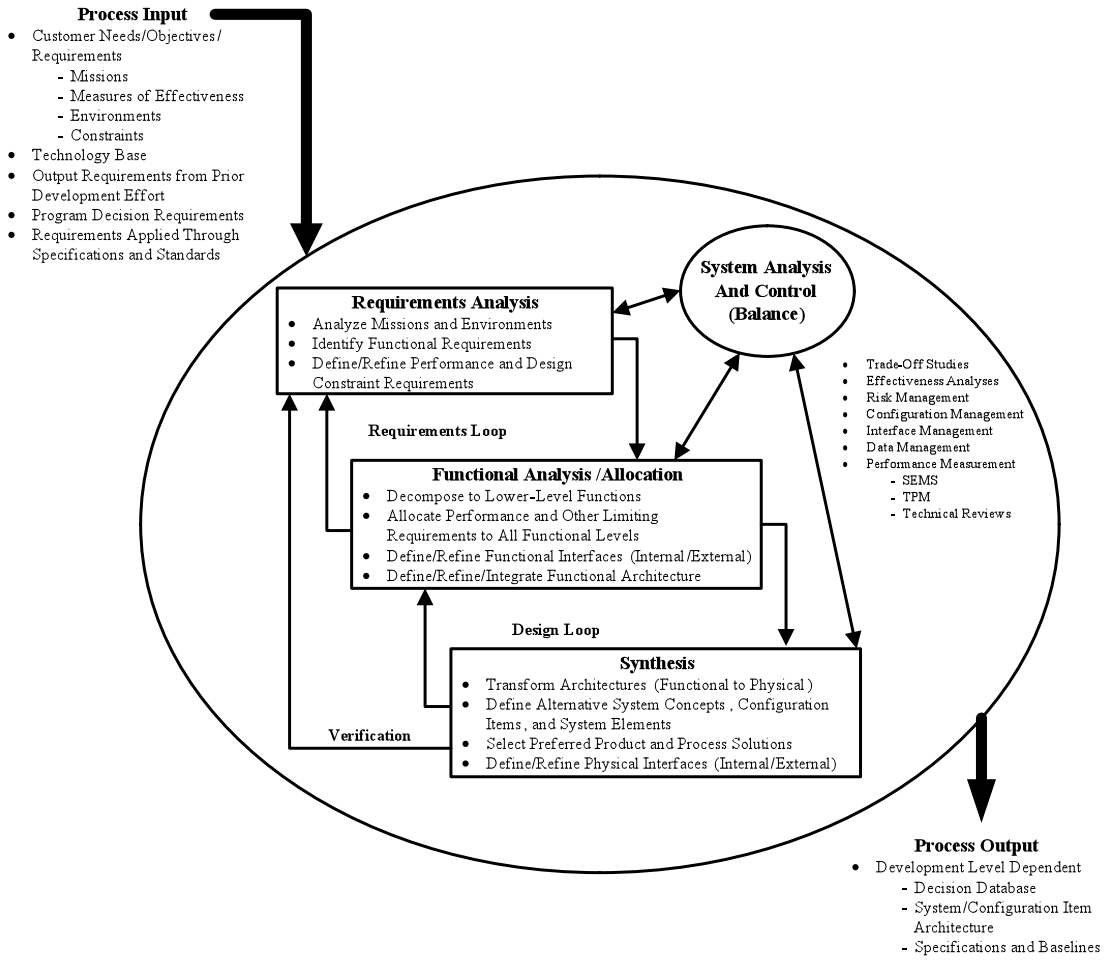

Figure 10 – Fundamental Systems Engineering Process

Figure 11 maps the need identification and solution response process (Figure 9) to the fundamental systems engineering process (Figure 10). The fundamental systems engineering's sub-process of *Process Input* maps to the need identification and solution response's sub-processes of *Identify Need* through *Develop Requirements*. The *Requirements Analysis* sub-process of each process map to each other. The fundamental systems engineering's sub-process of *System Analysis and Control (Balance)* maps to the *Formulate Solution Design Space* sub-process of the need identification and solution

response process. The *Functional Analysis And Allocation* sub-process from fundamental systems engineering process maps to the need identification and solution response's sub-process of *Perform Solution Trade Analysis.* The fundamental systems engineering process activity of *Synthesis* maps to the *Develop Solution* activity of the need identification and solution response process. The fundamental systems engineering's sub-process of *Verification* and *Process Outputs* map to the *Verify a*nd *Validate Solution* and *Present Results* activities of the need identification and solution response process.

Solution Resolution Process	SE Fundamental Activity
Identify Need thru Develop Requirements	Process Inputs
Requirement Analysis	Requirements Analysis
Formulate Solution Design Space	System Analysis And Control (Balance)
Perform Solution Trade Analysis	Functional Analysis And Allocation
Develop Solution	Synthesis
Verify And Validate Solution	Verification
Present Results	Process Output

Figure 11 – Need Identification and Solution Response Mapped To Fundamental Systems Engineering Process

1.2 Mission Critical Release Readiness Methodology Applicability

A mission critical release readiness methodology would be applicable to commercial and Department of Defense (DoD) projects developing software for mission critical systems. The mission critical release readiness methodology's applicable lifecycle stages would span from Concept through Support for a commercial project as shown in Figure 12 and Material Solution Analysis through Operations & Support for DoD projects as shown in Figure 13. The development of the mission critical release readiness methodology will focus on its applicability to DoD projects and the expansion to commercial projects will be left for future work.

Life Cycle Stages
Concept
Development
Production
Utilization
Support
Retirement

Figure 12 – Lifecycle Stages (from INCOSE Handbook)

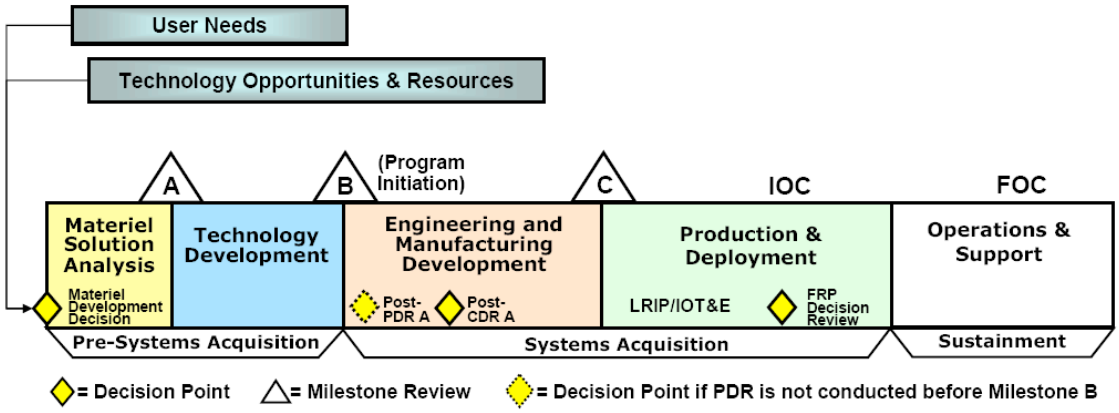

Figure 13 – The Defense Acquisition Management System (from DoD Instruction 5000.2, December 8, 2008)

Focusing development of the mission critical release readiness methodology on DoD projects simplifies calculations of project data as some factors such as time to market, market share, etc. will not be included.

1.3 Using Systems Engineering To Improve The Software Release Decision Process

Today's complex systems are becoming more and more integrated, as evidence by the growing field of System of Systems (SoS). Consequently, software is being integrated with other processors within a subsystem and across interfaces within the total system itself, increasing the: complexity of software; number of subsystems requiring software; and the schedule nuances of performing a software release; emphasis placed on integrated system releases of software; and the activities required to certify software for release. Applying systems engineering processes and tools should improve the software release decision process.

1.3.1 System of Systems Adds Complexity

System of systems, as the name implies, is a system comprised of other systems. A system composed of other systems adds additional complexity, performance constraints, and integration challenges. For instance, cars today may have 50 microprocessors controlling everything from the engine to the air bag [1]. Every microprocessor runs its own software and probably interfaces with additional microprocessors, driving additional complexity in the form of additional interfaces, timing issues, subsystems requiring software, and countless unnamed integration problems often resulting in reduced SoS reliability, safety, and higher risk. The Drive By Wire [2] technology for future cars, will only increase a car's complexity and integration challenges. Drive By Wire is similar to the Fly By Wire flight control system for aircraft where the mechanical linkage between the pilot controls and the surfaces are replaced with wires that send the commands from the controls to the surfaces in order to move the surfaces on the aircraft. In Drive By Wire, the mechanical linkage from the steering wheel is replaced by wires that send the signals needed to steer the car. In the past cars could be serviced by mechanically inclined individuals who did not mind getting their hands dirty. Today, one needs an advanced understanding of software products to service cars and the person making the car's software release decision probably has an advanced software or maybe even an advanced systems engineering degree assist them in making the release decision.

When the question is asked, "How much testing is required?" the answer is in the form of as much resources as are available [85]. However, it is neither economical nor

cost effective to test the software in every conceivable operational condition. Beyond the economical and cost constraints, it is just not possible [51].

1.3.2 Software Complexity

Not only are today's systems of systems becoming more complex, but the software for the SoS is becoming more complex almost on a daily basis. Software complexity can be measured in many ways [[87], [88], and [89]]. The number of Lines Of Code (LOC) influences the complexity calculations and by itself gives a relative idea of software complexity, the more lines of code, the more complex the software. Looking at operating systems line of code count shows an interesting trend [6]:

> Real systems show no signs of becoming less complex. In fact, they are becoming more complex faster and faster. Microsoft Windows is a poster child for this trend to complexity. Windows 3.1, released in 1992, had 3 million lines of code; Windows 95 has 15 million and Windows 98 has 18 million. The original Windows NT (also 1992) had 4 million lines of code; NT 4.0 (1996) has 16.5 million. In 1998, Windows NT 5.0 was estimated to have 20 million lines of code; by the time it was renamed Windows 2000 (in 1999) it had between 35 million and 60 million lines of code, depending on who you believe.

Windows Vista® reportedly contains 50 million lines of code [7]. The number of software releases for a product with 50 million lines of code has to be large.

Imagine performing only one software release for a product with 50 million lines of code.

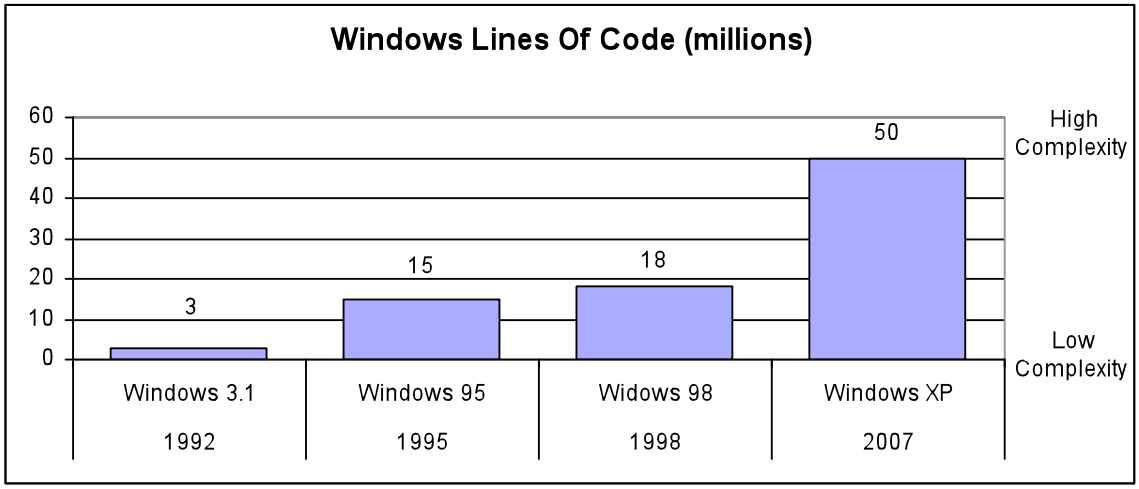

Figure 14 – Windows® Software Complexity Through Releases

1.3.3 Software Criticality

The very ubiquitousness of software makes it mission critical in many systems. Adding to the criticality of software is the sheer complexity of today's systems. The complexity and integration requirements of a SoS affects the system's software safety and impacts the software's criticality. As Leveson [3] points out:

> Today we are building systems – and using computers to control them – that have the potential for large-scale destruction of life and the environment: Even a single accident may be disastrous.

Software criticality is given a new meaning when one works with application software that could injure or kill someone. A software developer on the space shuttle summed it up by saying, "If the software isn't perfect, some of the people we go to meetings with might die." [21] Today's added complexity, additional requirements, and criticality of software, means the decision of when to release software is becoming as complex as the software itself. Add to the complexity the problem of multiple, integrated, software releases for a single system and the software release decision is not a decision that can be made easily. We will use systems engineering principles to develop an integrated software release decision methodology that considers the complexity, integrational aspects, and criticality of today's systems to realize a successful software release.

1.3.4 Software Releases

Many tools are available to assist in the planning of software releases. Mission critical software complicates software release planning. If the software is released, and does not function correctly, then the system may not support its intended purpose. Software that does not support its intended purpose could cause cost, schedule, or resource impacts to the developer or the customer of the software. Whereas various tools for planning software releases exist, the recommendation to release mission critical software is normally made by a team of experienced engineers, who have established that the software development and release processes were followed and that the software can support its intended purpose and may be partially based on the output of a tool – which

probably contains software itself. The team and tools provide recommendations to the software release decision maker, who has the ultimate say in when to release the software. In today's developing environment, a systems engineer probably assisted in developing the software development and release processes.

In order to reduce confusion, some definitions are required. A literature search revealed many terms and definitions related to software releases [8, 9, 10, 11]. We define production software release as a release to the end customer that has been validated and verified through modeling, simulation, and test to meet the requirements for that software release. An interim software release is defined as a release to a customer that is not fully verified or validated to all of the requirements. Customer, as used here, is defined as a user of the software. A customer could be internal or external to the company. An end customer is the customer that receives the production software after all verification and validation activities are complete.

Given today's integrated environment, releasing production software is an accomplishment in itself. With a production release, the design is complete, testing is complete, requirements are verified, outstanding problems are mitigated, contractual obligations have been met, the schedule no longer is a plan, it is the actuals for the program, and significant management oversight is provided, making the path for a production release familiar and the process well known. Accompanying a production release is a sense of accomplishment for a job well done and possibly the end of the program.

In today's integrated, complex systems, including systems of systems environment, it would be difficult, if not impossible, to proceed through a production

software program of any size with only a production software release. The complexity and integrated nature of systems of systems almost requires interim releases before the production release.

If the path to production release is well known and familiar, does it necessarily follow that the production software release path/process is adequate for interim software releases? The nature of an interim software release is that it contains partial functionality, occurs before the design is complete, and requirements may not be complete. Because of this nature, a production software release process where the design and testing are expected to be complete and requirements are expected to have been validated and verified, may not be the best process. Having a separate interim release path in the software release process insures the incomplete nature of the release is taken into consideration.

1.4 Dissertation Organization

This dissertation's chapters follow the fundamental systems engineering process, illustrated in Figure 4, sequentially. Chapter 1 introduces the problem. Chapter 2 expands on the problem and introduces the customer's needs. Chapter 3 expands on the customer needs and develops requirements for solving how to determine when to release software. Chapter 4 is the analysis of the requirements into candidate solutions. Chapter 5 is the development of the analytically chosen solution. Chapter 6 describes the development of the solution. Chapter 7 is verification and validation. Finally, chapter 8 summarizes the findings, presents the conclusions and future work.

The coverage of dissertation chapters versus the systems engineering fundamentals process and the need identification and problem resolution process is shown in Figure 15Figure 11. The first two sections of each chapter will discuss the systems engineering fundamentals and the systems engineering need identification and solution resolution sub-processes attributed to that particular chapter.

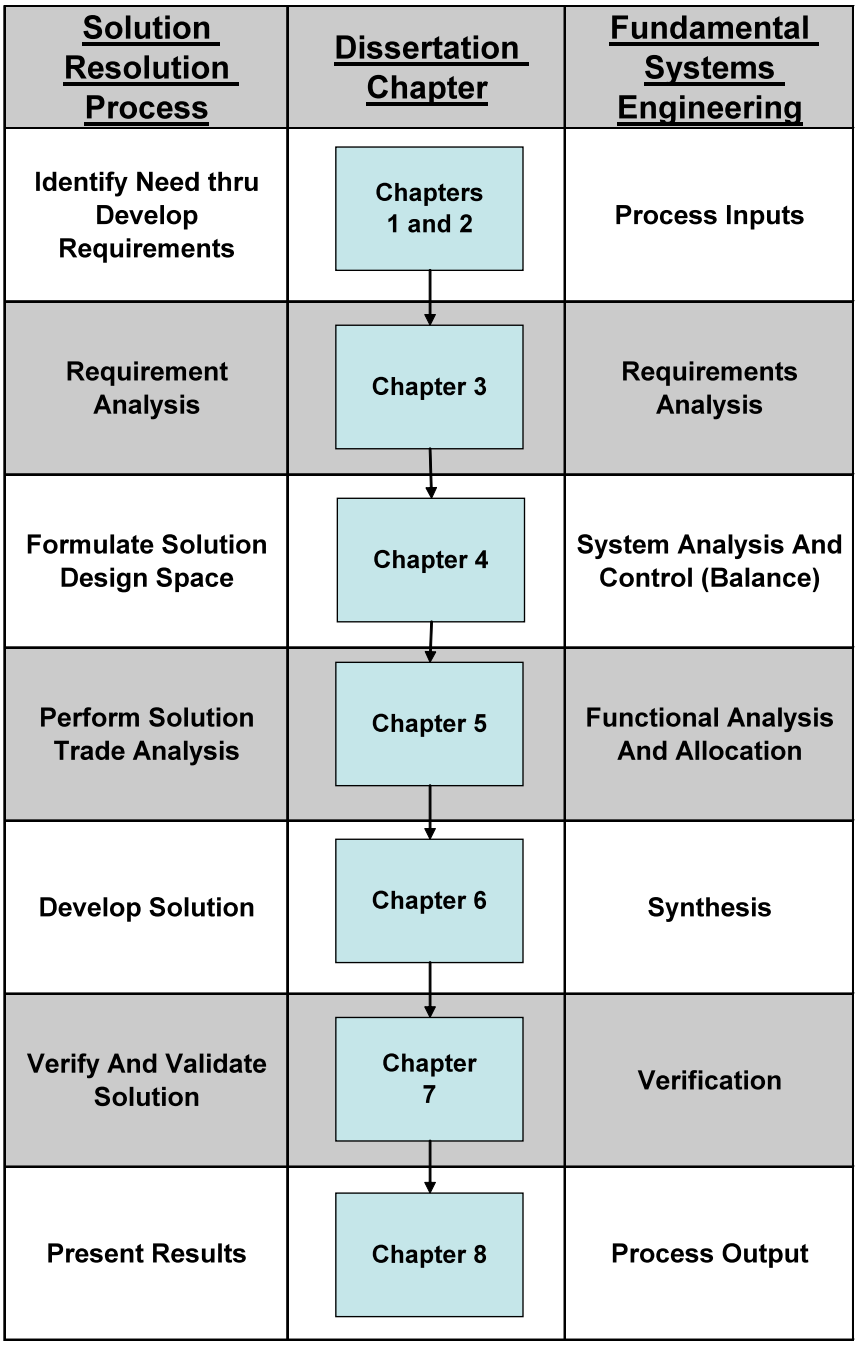

Figure 15 – Process Coverage By Dissertation Chapter

Chapter 2

CUSTOMER'S NEEDS/REQUIREMENTS

2.1 Process Inputs

The systems engineering fundamentals process requires inputs in the form of customer needs/objectives/requirements. The identification of the Customer Needs/Objectives/Requirements inputs to the systems engineering fundamentals was tailored in the need identification and solution process and identified as the *Research Customer Needs* sub-process. The customer is any user of the methodology, in our case the main customers are the software release manager, software manager, and the program manager. Other customers may use the data the methodology produces, but their needs are not seen as driving the behavior of the methodology at this time. In this chapter, we discus the *Process Inputs* sub-process by developing the needs of the customers and then derive requirements from those needs.

2.2 Research Customer Needs

In the systems engineering approach to need identification and solution resolution process, after a problem area has been researched and found to be lacking a solution, the next sub-process is the *Research Customer Needs*. In the *Research Customer Needs* sub-

process the research performed earlier in the process is analyzed to determine customer needs and requirements. The needs and requirements identified are then analyzed to derive requirements for the methodology. All needs and requirements are identified with a unique alpha-numeric sequence to allow tracking of the needs and requirements throughout the research.

In order to identify customer needs and requirements the problem area is examined further via literature searches and interviews with current customers of software release data. Then customer needs are identified, followed by the identification/derivation of the customer requirements. Finally, the identified and derived customer needs and requirements are analyzed to derive implementation requirements.

2.3 Why Release Software?

Given the complexity of today's systems and the complexity of software contained in systems, and the added complexity caused by mission critical software, why release software? The number one reason to release software is to make money. Software is considered intellectual property and if the value of the intellectual property is high, then the monetary value of the software increases, which means people (aka, customers) will pay more money for the software [38].

Another reason to release software is to fulfill a need. The need could be fulfilled without selling the software, as in the GNU Operating System, which is developed as freeware by a pool of people dedicated to developing and supporting a free operating system for anyone anywhere to use [39]. Or perhaps the software is released to fulfill the

customer's expectations regarding the intended purpose of the software – said in another way, a bug fix provided under the software's warranty period.

2.4 Customer Needs

The customers of the software release decision methodology have been identified as the software release manager, software manager, and the program manager. The main customer is the software release manager – the person actually making the decision to release the software. The software release manager's needs will have the largest impact on the methodology, but all of the customers needs will be considered.

The decision to release software in accordance with the program's development schedule normally falls to one person, the software release manager. The software release manager decides when to release the software and then receives program approval for releasing the software. The software release manager uses their understanding of program processes, program state, and personal history to choose to release software from several options provided by engineering, all the while trying to balance the intricacies of software releasing versus program goals. Given the importance and timing of when to release software, it is apparent that the decision of when to release software is not always cut and dry. What about a release option that was considered but never provided by engineering? What about another release option that engineering and the software release manager had not considered? Do the options provided follow the approved software release process? Are any requirements in danger of non-compliance due to the chosen release date? Does the chosen option support the intended purpose of the released software? Looking at the software release decision it is apparent that

choosing the release date is not an easy process. If a release readiness metric was to assist in the software release decision process, it is readily apparent that the software release manager would have several needs with regards to the release readiness metric.

The software manager manages the software design and development. The software manager needs a current status of the software release decision and factors affecting the release so the software team can adequately plan their resources to insure the success of the software release. In some organizations, the software manager may also be the software release manager.

The program manager is the ultimate decision authority on releasing changes to the system and therefore on releasing software. When dealing with complex systems the program may consist of various tiers and each tier may have a software release manager that manages the software release decisions for that tier. All the software and hardware changes roll up to the top level tier and the program manager decides whether to proceed with the system release. Figure 16 illustrates a tiered organization with multiple software release managers and a program manager at tier 1.

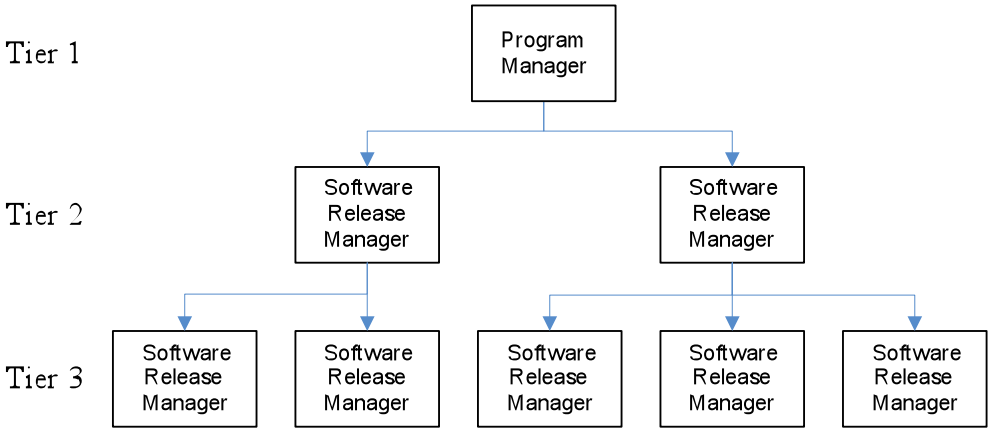

Figure 16 – Tiered Organization

The customer needs were derived through a modified systems engineering requirements elicitation process. The needs elicitation process involved researching the software release decision process, analyzing the software release, software, and program manager job duties through research and current job listings, and interviewing managers. The needs elicitation process is shown in Figure 17.

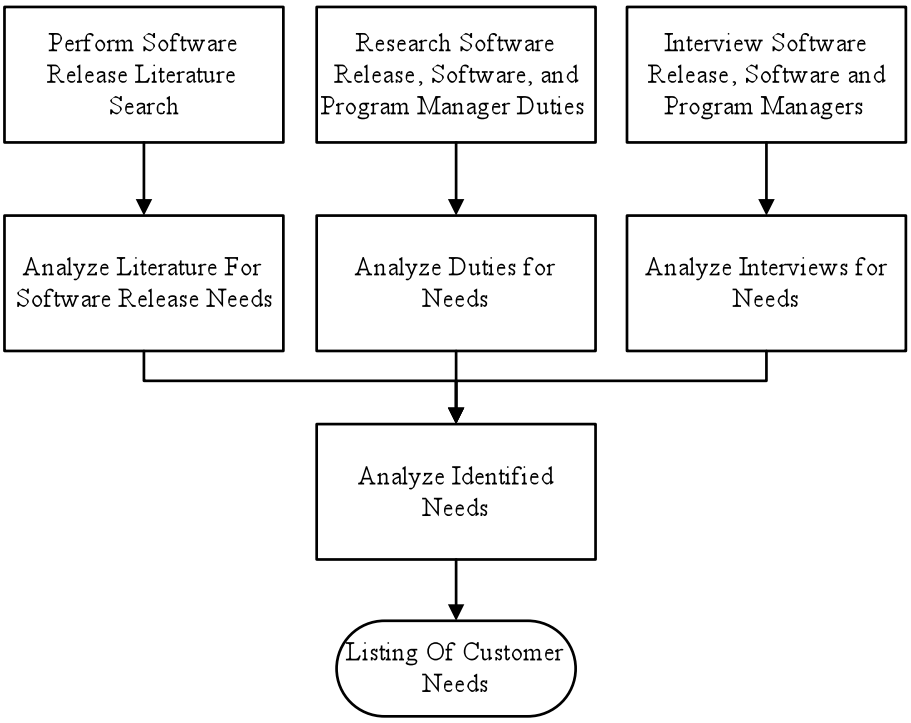

Figure 17 – Needs Elicitation Process

The needs elicited through the needs elicitation process are shown in Table 1, along with their unique program identifier. The customer's needs are used as the basis for the methodology requirements in the next section. The customer's needs and methodology requirements are then analyzed for derived implementation requirements in a later section. The customer's needs are further elicited in the following paragraphs.

Customer, as used in the following paragraphs, refers to a user of the methodology or the data produced by the methodology.

Table 1 – Customer Needs For Software Release Decision Methodology

Identifier	Statement of Need
N1	The customer needs to have timely access to decision analysis support results.
N2	The customer needs concurrence that the approved software release process is being followed during the release of the current software version.
N3	The customer needs accurate information to assist in the release decision.
N4	The customer needs some measure of the ability of the software to support its intended purpose based on the chosen release option.
N5	The customer needs to know what effect the software has on requirements verification and validation.
N6	The customer needs to know if additional processing/testing is required when releasing the software.
N7	The customer needs to know the costs involved with releasing the software.
N8	The customer needs to know the software is secure.
N9	Software needs to be stable.

Assuming the program timeline is changing, which if a decision is needed on releasing software would be a safe assumption, the customer needs timely access to data to assist him/her in the software release decision process. If the program is in flux and changes are happening, it would not be a good time to take 3 or 4 days to analyze the software release problem and then finally present analytical results that might assist the customer in their decision making. Every day it takes to decide when to release the software could be a days slip in the program and most program managers will not hesitate

to make a decision, without any supporting data analysis, on replacing the software release decision maker for taking too long to decide whether to release the software.

Another need of the customer is to the need to follow company processes regarding releasing software. Today's software development environment almost requires a company to be Capability Maturity Model Integration (CMMI) certified to manage the complexities found in developing software [41]. If a particular software release path does not follow the company process, it is important for the software release decision maker and program management to know the process was not followed and be able to evaluate the release for problems.

Timely information and meeting company processes are important needs of the customer, but if the data provided meets the processes and is timely, but not accurate, other problems arise. Decision analysis using bad data may result in a bad decision. The customer needs accurate information to assist in the release decision making process.

In order to make an informed decision on releasing software, the customer needs to have some measure of the software's ability to support its intended purpose when it is released. The software's intended purpose may change during its developmental life cycle. Software may be in development to control a car, but that does not mean the software must be used for that purpose for every release. Perhaps the software is needed to support integration of dashboard displays in the laboratory. In this case, the software team may release a test only version of software that is used to support the laboratory testing. The intended purpose is support integration of the dashboard displays; it does not have to be capable of controlling an automobile just that it is capable of supporting the dashboard displays testing in the laboratory. Making a decision to release software with

some measure of the software's goodness and/or its ability to perform the function it is intended is a definite need of the customer.

The needs of the customer do not end when the software is released. In order to make an informed decision, the customer needs to know what effect the software release will have on requirements Verification and Validation (V&V). When developing complex systems, it is possible that releasing software that will support its intended purpose, but not support requirement V&V. The affect on V&V either as part of the software's intended purpose now or a future purpose is a signification need of the customer.

Deciding when to release software would be a fairly easy task assuming the software was released the same way every time. But an organization may have multiple release options: lab, integration, test, production, etc. and the releases may require different processing and/or testing based on the release option. The customer needs to know not only what the intended purpose of the release option is, but also, any special requirements placed on the software release based on the release option.

Should the customer consider the cost of releasing software? One would think that the cost to release software would be fairly well fixed and known, right. It should be known how long it takes to build the software, release the software from development to test, test the software, and then release the software for its intended purpose. Barring any major findings during the process, it should proceed at a fairly consistent basis allowing the cost of release to be known and tracked by the customer.

Today's environment forces a need for the customer to know if the software released will be secure. Similar to a chain breaking at its weakest point, a system's security is only as secure as its weakest software point [4].

Another need of the customer is to know the stability of the software [43]. Experience has shown that end users are not happy when the software provided is unstable. The customer needs to understand the stability of the software about to be released.

2.5 Customer Needs to Requirements

Using the customer needs as the basis, methodology requirements can be derived from the needs and the methodology requirements derived are shown in Table 2.

Table 2 – Methodology Requirements

Parent Ident	Req ID	Requirement Text
N1	R1	Once the methodology has been initialized, it shall take no more than 4 hours for the methodology to produce decision analysis support results.
N2	R2	The methodology shall indicate whether the software release follows the software release process.
N3	R3	The methodology shall provide the date the methodology analyzed the software release.
N4	R4	The methodology shall provide a qualitative measure on the software release's ability to perform its intended purpose.
N5	R5	Software releases containing severity 1 or 2 problem reports affecting requirement V&V shall be identified.
N6	R6	The methodology shall list software releases requiring additional processing or testing as part of the release process.
N7	R7	The methodology shall permit the software release manager to track the cost of the software release.
N8	R8	The release options requiring security processing shall be identified
N9	R9	The released software shall be capable of supporting its intended purpose.

2.6 Summary of Customer's Needs/Requirements

Nine customer needs regarding software release decision making have been identified using systems engineering processes and following the need identification and problem resolution process. Nine corresponding methodology requirements have been derived from the nine needs. The next step, according to the system engineering fundamental and the need identification and problem resolution process processes, is to perform requirements analysis. The requirements analysis is shown in the next chapter.

Chapter 3

REQUIREMENTS DEVELOPMENT

3.1 Requirements Analysis

In the previous chapter, the customer needs were researched and identified, methodology requirements were derived from the needs, and these needs and requirements now become the inputs to the Requirements Analysis sub-process. The inputs will be analyzed to develop requirements that envelop the minimum performance objectives, while detailing the constraints of the system under development. Typically the systems engineering fundamentals process is iterative with the process being performed at every level of the organization until all requirements are identified and flowed down. In this case, there is only one level and the process will only be applied once and not iteratively.

3.2 Requirements Analysis

The need identification and problem resolution process sub-process of Requirements Analysis is similar to Requirements Analysis sub-process of the systems engineering fundamentals process. The customer needs and methodology requirements are used as inputs to the analysis needed to develop requirements for implementing the

methodology. The developed requirements will determine the minimum objectives and detail any constraints of the decision process. The output of this sub-process shall be a set of derived requirements, verification methods for the derived requirements, performance measures for the methodology, and any identified methodology constraints.

Requirements analysis at this phase is not meant to generate all the methodology requirements, but to produce an understanding of the methodology and its objectives and constraints to allow the process to continue under reduced risk [44].

3.3 Verifiability of Requirements

One aspect of a good requirement is that the requirement must be verifiable [45], therefore, the first step in the requirements analysis process performed was analyzing the customer requirements for verifiability. The verifiability analysis was performed by examining the requirement and determining if a verification method could be attributed to that requirement. A requirement can be verified through one of four possible methods: analysis, demonstration, inspection, or test. Only two verification methods were identified for verifying the methodology requirements test and inspection. The unique development environment for the methodology does not lend itself to a demonstration verification method and the current requirements have not required analysis as a method of verification at this point in the process. The methodology requirements, verification methods, and verification comments are shown in Table 3.

Table 3 – Methodology Requirements Verification Methods

Parent Ident	Req ID	Requirement Text	Verification Method	Verification Comments
N1	R1	Once the methodology has been initialized, it shall take no more than 4 hours for the methodology to produce decision analysis support results.	Test	Initialized is defined as the methodology has been set up to run within the constraints of the release process.
N2	R2	The methodology shall indicate whether the software release follows the software release process.	Inspection	
N3	R3	The methodology shall provide the date the methodology analyzed the software release.	Inspection	Possible derived requirements.
N4	R4	The methodology shall provide a qualitative measure on the software release's ability to perform its intended purpose.	Inspection	Base measure on current problem reports and requirements data. Severity 1/2 and/or large numbers of open problem reports would indicate less ability to support its intended purpose.
N5	R5	Software releases containing severity 1 or 2 problem reports affecting requirement V&V shall be identified.	Inspection	
N6	R6	The methodology shall list software releases requiring additional processing or testing as part of the release process.	Inspection	
N7	R7	The methodology shall permit the software release manager to track the cost of the software release.	Inspection	
N8	R8	The release options requiring security processing shall be identified	Inspection	Internal releases may not require the security processing options.
N9	R9	The released software shall be capable of supporting its intended purpose.	Inspection	Work arounds are permitted as long as software supports its intended purpose.

3.4 Derived Requirements

The methodology requirements were analyzed for clarifications and/or implementation specific requirements. If a clarification of a methodology requirement was required, a new derived requirement was developed. The list of derived requirements is shown in Table 4.

Table 4 – Derived Requirements

Parent Identifier	Req ID	Requirement Text	Verification Method
R3	DR1	The methodology shall use contractual obligations as an input.	Inspection
R3	DR2	The methodology shall use the current software release plan as an input.	Inspection
R3	DR3	The methodology shall use the current problem reports as an input.	Inspection
R3	DR4	The methodology shall use the current software requirements.	Inspection
R3	DR5	The methodology shall use current resource availability as an input.	Inspection
R3	DR6	The methodology shall use the current software release process as an input.	Inspection
R7	DR7	The methodology shall use the cost of releasing software as an input.	Inspection

3.4.1 Minimum Performance

A Technical Performance Measurement (TPM) provides the capability for measuring attributes of a system to determine how well the system is meeting specified requirements [46]. Due to the time critical nature of the software release decision, the time it takes the methodology to produce decision analysis support results will be the technical performance measurement used to determine how well the methodology is meeting its requirements. The technical performance measurement is shown below:

> TPM1 – The mission critical release readiness methodology shall provide analytically based, release decision support to the user with an objective of less than 1 hour and a not to exceed threshold of 4 hours from the time of initialization, after methodology set-up.

Initialization is defined as the methodology has been given the current set of inputs needed to analyze the software release under consideration. TPM1 is meant to measure the time it takes for the methodology to analyze a specific software release. It is not meant to include the time spent gaining access to required inputs or the time required to format inputs for the methodology's use.

3.4.2 Software Release Decision Methodology Constraints

At this point, the software release decision methodology consists of customer needs, methodology requirements, derived implementation requirements, verification methodologies for the requirements, and a technical performance measure. Analysis of

the software release decision methodology does not indicate any methodology constraints at this time.

3.5 Summary of Requirements Development

In this chapter, the nine methodology requirements were analyzed for verifiability, and completeness. All the methodology requirements were identified with a verification method. Additionally, seven derived requirements were developed to provide clarification of the methodology inputs from the methodology requirements and identified with a verification method. No design constraints were identified at this time and one technical performance measurement was developed regarding the time required for the methodology to provide decision analysis support results.

Chapter 4

METHODOLOGY DESIGN SPACE AND ANALYSIS

4.1 Systems Analysis and Control

The following activities are of the *Systems Analysis and Control* sub-process as described in the Systems Engineering Fundamentals guide [17]:

> "*Systems Analysis and Control* include technical management activities required to measure progress, evaluate and select alternatives, and document data and decisions. These activities apply to all steps of the systems engineering process. System analysis activities include trade-off studies, effectiveness analyses, and design analyses. They evaluate alternative approaches to satisfy technical requirements and program objectives, and provide a rigorous quantitative basis for selecting performance, functional, and design requirements. Tools used to provide input to analysis activities include modeling, simulation, experimentation, and test.
> Control activities include risk management, configuration management, data management, and performance-based progress measurement including event-based scheduling, Technical Performance Measurement (TPM), and technical reviews."

In this case, the *Systems Analysis and Control* sub-process activities correspond to the technical management activities for evaluating and selecting alternative methodologies for the system engineering the mission critical software release decision.

4.2 Formulate Solution Design Space

With the customer's needs, methodology requirements, and derived requirements identified and developed, now the need identification and problem resolution process executes the *Formulate Solution Design Space* sub-process. This sub-process is used to formulate multiple feasible design solutions and then map those solutions to provide a visual representation of the total design space for the system under analysis.

4.3 Candidate Methodology Designs

In order to assist in bounding the how to decide when to release software methodologies design space, research of current software release methodologies was performed. Once the research was complete, a brainstorming/white boarding activity was performed on possible methodology designs in order to work outside the comfortable "box" [48] and insure the design space adequately represented possible designs.

4.3.1 Software Release Methodology Research

A literature survey of software related books was performed as a starting point for the research on software release methodologies [[3], [6], [7], [25], [29], [38], [58], [66], and [67]]. The literature survey found the majority of the books did not mention software release, much less a software release decision methodology, but one book did, Software Release Methodology. It does not contain any references and although it discusses many aspects of releasing software, only generic philosophical release methodologies are presented with no consideration for resources (both laboratories and personnel) and risk.

The next step in researching software release methodologies was to perform web searches. The search sites of Google and Google Scholar and websites for IEEE and Crosstalk were all searched using combinations of software release methodology, release methodology, and software release. As the search returned items of interests, the items were archived for later, more rigorous research. Also, as items of interest were returned, the searches were expanded with software release methodology processes and/or tools.

Software release was also used as search criteria in the ProQuest Dissertations & Theses databases. The ProQuest search returned 34 items, but only nine appeared to be related to software release when examined closer. The nine items were identified for later, more rigorous research.

The most significant software release methodology processes and/or tools returned from the web and ProQuest searches are shown in a later section entitled Formal Decision Methodologies.

4.3.2 Software Release Methodology Brainstorming/White Boarding

In addition to the literary survey and web searches, software methodologies were the topic for several brainstorming/white boarding activities. During brainstorming/white boarding all ideas, no matter how seemingly off subject, are thrown out and documented. Then the ideas are sorted for relevance and the relevant ideas are further refined into a listing of ideas to consider. Brainstorming/white boarding produced several relevant software development and software planning methodologies (Agile, Program Evaluation and Review Technique (PERT) and the COnstrunctive COst Model (COCOMO)), but no additional formal software release methodologies. However, several informal software

methodologies were identified during the brainstorming/white boarding activities. Although the informal software release methodologies are included in the design space, their informal nature limits the amount of reference material available on the methodologies and therefore limits their consideration as candidates for a software release decision methodology. The informal software release methodologies are identified in the following section.

4.3.3 Informal Software Release Methodologies

Several informal software release methodologies have been used to assist in deciding to release software. These methodologies are listed as informal due to being based in the qualitative area of decision making and because of their lack of quantitative analysis to support the decision making. These informal decision methodologies were all identified during the brainstorming/white boarding activities of the previous section.

4.3.3.1 Bunch Of Guys Sitting Around a Table

Bunch Of Guys Sitting Around a Table (BOGSAT) is not only a software release decision methodology, but a common decision making methodology. The decision maker (the Software Release Manager in the case under study) gathers the Subject Matter Experts (SMEs) around a table and asks for inputs regarding releasing the software. Once the inputs have been received, the decision maker decides whether to release the software or not.

The major problem with BOGSAT as a software release decision methodology is its reliance on people and their current interpretation of the software development

lifecycle and release process. Also, BOGSAT involves multiple people, but are the proper SMEs sitting around the table? Have all aspects of the software release decision been considered? Is the BOGSAT methodology working with the latest data? Are all the problem reports being considered? Are requirements being considered? What is the definition of consensus to release? Etc.

4.3.3.2 Squeaky Wheel Gets The Grease

Deciding to release software is sometimes dominated by the person or group that complains the loudest, hence, the squeaky wheel gets the grease [49]. In this case the grease is a software release benefitting the person or group that complained the loudest. Besides being a one-sided decision making process, deciding when to release by the squeaky wheel methodology means the software release decision benefits mainly the squeaky wheel and not the whole program. Releasing software to benefit the minority could have unattended effects on the rest of the team and adversely affect the program schedule. The squeaky wheel methodology also tends to silo the various groups on the team as they learn to squeak louder than the other groups to the benefit of their group.

4.3.3.3 Releasing Per the Plan

Releasing per the plan is a release methodology where the original program plan with its schedule of releases is used as the sole means of deciding when to release the software. Just taking into consideration one factor, the original plan, for deciding when to release software is an archaic methodology, especially with today's complex software and systems. The plan developed at the start of the program or even one developed last week, does not have the most up to date and current data to determine whether the

software should be released. Releasing per the plan is sticking to a plan, because it is the plan and therefore it must be correct and is of little benefit to any program.

4.3.3.4 Schedule-Driven Development Program

The schedule-driven development program is one of the more basic approaches, given a schedule; release the software as scheduled. True schedule-driven programs are rare [50]. If a schedule-driven program is properly managed, and the reasons for using the methodology have been communicated to the team, the schedule-driven methodology, if properly resourced, can deliver the same capability before the standard approach. However, schedule-driven development is not a software release methodology; it is a development process and as a development process, it is not considered any further.

4.3.4 Formal Software Release Methodologies

Researching software release methodologies provides insight into software releasing decision making and the current trends of software release methodlogies. Most of these formal methodologies have some quantitative analysis built into their methodologies. The formal software release methodologies researched are listed in the following sections.

4.3.4.1 6C

The 6C methodology combines a software reliability growth model with other quality metrics, such as risk and reliability, to determine when to release the software. The 6c's are [61]:

1. Consider the target reliability.

2. Collect and model failure date.

3. Classify the defect data and run a SRGM fitness test.

4. Capture Trend.

5. Certify by considering parameters such as risk.

6. Consolidate different solution alternatives.

The first C – Consider target reliability is accomplished by first calculating the target reliability for the project $\lambda_{(final)}$, using the following equations:

$$\lambda_{(final)} = \lambda_{(overall)} - \lambda_{other}$$

$$\lambda_{(overall)} = (1 - A(t)) / (t_m \times A(t))$$

Where:

$\lambda_{(final)}$ = project specific target reliability
$\lambda_{(overall)}$ = overall reliability
$\lambda_{(final)}$ = expected reliability of hardware and acquired software components
A(t) = desired availability of the system over a period of time
t_m = average downtime per failure

The second C – Collect and model failure data is accomplished by collecting and analyzing failure/defect data. The third C – Classify the defect data and run a SRGM fitness test classifies the defect data collected and then runs a fitness to be able to estimate the remaining defects. The fourth C – Capture trend performs trend analysis on

the failure data to track the project's relation to the target level reliability. The fifth C – Certify by considering parameters such as risk analyzes when the target reliability is met and with what level of risk. The sixth C – Consolidate different solution alternatives provides and opportunity for the decision maker to consider the data collected in the preceding steps and the project dependent factors to make final decisions.

The 6C methodology uses reliability as the major release decision factor. While acknowledging other project dependent factors exist, they other factors do not factor into the methodology other than consideration by the decision make. Although reliability is important, there are other needs of the software release manager and 6C does not meet all the needs.

4.3.4.2 Agile Software Development

Many different methodologies fall under Agile Software Development: Dynamic Systems Development Method (DSDM), Scrum, Adaptive Software Development (ASD), and XP [62]. Agile methodologies are used to respond quickly, with the user in mind, while developing software in small increments normally developed via customer developed use cases, with the software being built nearly every day and the software released every month or so [63].

Agile software development was identified in several of the searches and the brainstorming/white boarding activity. Agile is a software development methodology, not a software release methodology. The agile software development methodology does include references to software releases, but not in any great detail. While researching agile software development, several aspects of software release from different agile

methodologies were identified: multiple frequent builds; monthly (or so) software releases; and passing all release testing with no failures, before releasing the software. Due to its prevalence in the brainstorming/white boarding activity and the literature search, agile software development will be included in the release methodology analysis to insure completeness and to compare and contrast to a software release methodology.

4.3.4.3 COnstrunctive COst Model (COCOMO)

The COnstrunctive COst Model (COCOMO) was originally developed in 1981 by Dr. Barry Boehm [59]. The COCOMO user guide has the following description of the model:

> COCOMO (COnstructive COst MOdel) is a screen-oriented, interactive software package that assists in budgetary planning and schedule estimation of a software development project. Through the flexibility of COCOMO, a software project manager (or team leader) can develop a model (or multiple models) of projects in order to identify potential problems in resources, personnel, budgets, and schedules both before and while the potential software package is being developed [55].

COCOMO's underlying general cost model is [70]:

$$PM = Ax\left(\sum Size\right)^{\sum B} x\prod(EM)$$

where,

PM = person months.
A = calibration factor.
Size = measure(s) of functional size of a software module that has an additive effect on software development effort.
B = scale factor(s) that has an exponential or nonlinear effect on software development effort.

EM = effort multipliers that influence software development effort.

The brainstorming/white boarding activity and the literature search both identified COCOMO as a possible software release methodology. Further research indicates COCOMO is for software project planning and not software releasing. Due to its prevalence in the brainstorming/white boarding activity and the literature search, COCOMO will be included in the release methodology analysis to insure completeness and to compare and contrast to a software release methodology.

4.3.4.4 EVOLVE

EVOLVE is a methodology for planning incremental software releases using an iterative, genetic based algorithm [60]. EVOLVE was developed for use in planning incremental software developments using the requirements, constraints, and user based priorities. The EVOLVE methodology is meant to be applied after every iteration to arrive at the optimal release solution.

The following presents a summary of the genetic algorithm developed for use in EVOLVE [60].

Input:

Sseed = Initial seed solution

m = population size

cr = crossover rate

mr = mutation rate

Output:

The solution with the highest fitness score from the final population

Variables:

Sn =A Solution

P =current Population as a set of (Solution, fitness score) pairs = {(S1,v1), ,(S2, v2)….(Sm,vm)}

Sparent1 = first parent selected for crossover

Sparent2 = second parent selected for crossover

SOffspring = result from crossover/ mutation operation

Functions:

NewPopulation(Sseed,m): Sseed→P , Returns a new population of size m.

Evaluate(S) provides a fitness score for a given solution, S.

Select(P) chooses from population P, based on fitness score, a parent for the crossover operation.

Crossover(Si,Sj,cr) performs crossover of solutions Si and Sj at crossover rate cr.

Mutation(Si, mr) performs mutation on solution Si at mutation rate mr.

IsValid(Si) checks validity of solution Si against the user-defined constrraints

BackTrack(Soffspring) = proprietary backtracking operation on a given solution. This backtracks towards the first parent until a valid solution is created or a user-defined number of backtrack operations is reached.

Cull(P) removes the (m+1)th ranked solution from the population, P.

CheckTermination() is a Boolean function which checks if the user's terminating conditions have been met. This may be when a number of optimizations have been completed, when there has been no change in the best fitness score over a given number of optimizations, a given time has elapsed or the user has

interrupted the optimization.

Max(P) returns the solution in population P that has the highest fitness score.

Algorithm:

```
BEGIN
    P := NewPopulation(seed);
    TerminateFlag := FALSE;
    WHILE NOT (TerminateFlag)
        BEGIN
            Sparent1 := Select(P);
            Sparent2 := Select(P/ Sparent1);
            SOffspring := Crossover(Sparent1, Sparent2, cr);
            SOffspring := Mutation(SOffspring, mr);
            If NOT IsValid(SOffspring) THEN BackTrack(SOffspring);
            IF IsValid(SOffspring)
                BEGIN
                    P := P ∪ {(SOffspring, Evaluate(Soffspring)}};
                    Cull(P);
                END;
            TerminateFlag = CheckTermination();
        END;
    RETURN(Max(P));
END.
```

EVOLVE is a planning tool for determining the functions to include in the next software release increment, it is not a software release methodology. EVOLVE was included in the release methodology analysis to insure completeness and to compare and contrast to a software release methodology.

4.3.4.5 Program Evaluation and Review Technique

Developed for the Navy's Polaris Project, the Program Evaluation and Review Technique (PERT), is a network model that uses task timing to develop the critical path of the program [56]. With the critical path identified, a program can easily identify where to spend resources to reduce the program's risk and required time.

PERT uses three estimations on activity completion times: optimistic time – the minimum time required to complete the activity; most likely time – the time with the highest probability of activity completion; and pessimistic time – the longest time required to complete an activity. PERT calculates an expected time for activity completion and uses that to calculate the programs critical path. Expect time is calculated using the three time estimates as follows:

Expected Time = (Optimistic Time + 4 x Most Likely Time + Pessimistic Time) / 6

PERT only deals with activity timing and scheduling, which does not meet all the needs for software releasing.

4.3.4.6 ShipIt

ShipIt is a calculated software release readiness metric that incorporates the completion status of the various software development stages and multiple relevant metrics related to producing software [69]. The metric is calculated in a 0 to 1 scale, with 1 indicating the software is complete and ready to release. ShipIt uses seven major components in factoring the metric:

1. Requirement Analysis Design Stage
2. Coding
3. Testing
4. Quality assurance
5. Manuals and Documentation
6. Supervision
7. Support

Of the seven components, five (Requirement Analysis Design Stage, Coding, Testing, Manuals and Documentation, and Supervision) are computed strictly on a percentage complete basis and two of the components (Quality Assurance and Support) are computed using other metrics. The ShipIt coefficient calculation is shown below:

$$ShipIT = [(WRAD \times RAD) + (WCODE \times CODE) + (WTEST \times TEST) + (WQA \times QA) + (WMD \times MD) + (WSV \times SV) + (WSP \times SP)] / 100$$

Where:

WRAD, WCODE, WTEST, WQA, WMD, WSV, WSP Є [0, 100]
RAD, CODE, TEST, QA, MD, SV, SP Є [0, 1].
WRAD + WCODE + WTEST + WQA + WMD + WSV + WSP = 100

ShipIt computes a zero to one metric that can be used to asses the status of the software for releasing. Although ShipIt considers several factors affecting the ability to release software, the complete listing of software release manager needs are not accounted for by ShipIt. Providing a metric or coefficient to indicate the status of the software releasability has considerable merit and will be considered as the software release decision methodology is developed.

4.3.4.7 Stopping Rule or Software Reliability Growth Model

Deciding when to release software using the stopping rule problem is similar to the Software Reliability Growth Model (SRGM) process. The SRGM process minimizes the total average cost to determine the optimal time to release software once the software is through development, testing, and error correction [52]. The stopping rule adds fault corrections to the SRGM process and the software is released at the optimal time determined by analyzing the costs involved to continue testing, versus the benefits of releasing the software [53]. Both processes may apply various SRGMs: Jelinski-Moranda; Goel-Okumoto; exponential; modified exponential; or S-shaped distribution. The Goel-Okumoto model is shown here and its use to determine software reliability. The derived economic cost model as it applies to the stopping model with fault corrections is given by [[53], [71]]:

$$E = C_1(t_r) + C_2 m_D(t_r) + C_3 m_C(t_r) + C_4(a - m_C(t_r)) + C_5\left(a \times b \times \left(1 + \frac{\log(m_C(t))}{a}\right)\right)\phi l$$

Where,

$m_d(t_r)$ is expected number of faults detected at time t_r.
$m_c(t_r)$ is expected number of faults corrected at time t_r.
ϕ is the expected execution time of the software release per field site.
l is the number of field sites.
C_1 is the cost of testing activities.
C_2 is the cost of resolving a failure.
C_3 is the cost of removing a fault and verifying the failure no longer occurs.
C_4 is the cost of fixing a fault which causes failures in the operational phase.
C_5 is the cost to customer operations in the field.
a is the expected number of faults detected.
b is related to the reliability growth rate of the testing process.

The stopping rule method for software release is based on the economic benefits of continuing software testing versus stopping testing to decide when to release software. Although the stopping rule factors in costs with the testing, the lack of consideration for the other software release manager needs, limits the stopping rule methods use as the MCRRM.

4.3.4.8 System Evaluation and Estimation of Resources – Software Estimating Model

The System Evaluation and Estimation of Resources – Software Estimating Model (SEER-SEM) is a software project estimation model that uses the output of several models to provide levels of: effort; duration; staffing; and software defects for a given software project [54]. SEER-SEM estimates software size as follows:

$$S_e = \text{NewSize} + \text{ExistingSize} \times (0.4 \times \text{Redesign} + 0.25 \times \text{Reimpl} = 0.35 \times \text{Retest})$$

SEER-SEM also translates function-based sizing metrics into Unadjusted Function Points (UFP) and converts them into software size as:

$$S_e = Lx \times (AdjFactor \times UFP)^{(Entropy/1.2)}$$

Where,

Lx is a language dependent expansion factor.
AdjFactor considers phase at estimation, operating environment, application type and complexity.
Entropy depends upon the type of software being developed.

SEER-SEM also performs effort and duration calculations as:

$$K = D^{0.4}(S_e/C_{te})^{1.2}$$

$$t_d = D^{-0.2}(S_e/C_{te})^{0.4}$$

Where,

K is basic effort.
t_d is duration.
S_e is effective size.
$C_{te}$is effective technology.
D is staffing complexity.

The brainstorming/white boarding activity identified SEER-SEM as a possible software release methodology. Further research indicates SEER-SEM is for software project planning and not software releasing. SEER-SEM was included in the release methodology analysis to insure completeness and to compare and contrast to a software release methodology.

4.3.4.9 Zero-Failure Method

The zero-failure method of software release specifies a number of testing hours that must be completed with zero failures found before releasing the software. If a failure is found during the test time, the planned release is cancelled and testing continues [68]. The basic assumption of this software release methodology is that zero failures over a specific, calculated, period of testing indicates a lower probability of additional failures in the software. The zero-failure method is based on the exponential model problem rate shown below:

$$p(t) = axe^{-bt}$$

To calculate the hours required with zero failures, three inputs are required: the t projected average number of failures received by the customer, the total number of failures detected during test, and the total testing hours up to the last failure. These three inputs are used in the zero-failure calculation as below:

$$Zero-FailTestHours = \left[\frac{\ln\left(\frac{Customer\,Received\,Problems}{0.5 + Customer\,Received\,Problems} \right)}{\ln\left(\frac{0.5 + Customer\,Received\,Problems}{TestFailures + Customer\,Received\,Problems} \right)} \right] xTestingHoursUpToLastFailure$$

The zero-failure method for software release depends solely on the number of hours with no failures as a decision method of when to release software. The reliance on finding problems, no matter how severe the problem or what the effect, and the lack of

consideration for the other software release manager needs, limits the zero-failure methods use as the MCRRM. As the author of the zero-failure method states [68]:

> "Finally, no one tool or method should be relied on to arbitrarily make the final determination of whether a software product should be released. Other factors may be at least as important to achieve the ultimate goal of quality assurance: total customer satisfaction."

4.4 Design Space Mapping Of Researched Software Release Methodologies

The candidate software release methodologies identified and analyzed in the list each have their own strengths and weaknesses and the methodologies may not even be applicable to deciding when to release mission critical software. In the problem under consideration, releasing mission critical software from test for its intended purpose, analyzing each identified methodology against the requirements would indicate the researched methodologies applicability to deciding whether to release mission critical software.

Analyzing each methodology to the requirements was accomplished by reviewing the methodology and deciding whether the methodology is a candidate to meet the requirement. The requirements and derived requirements were then used as the X and Y axis on a XY chart and the methodologies were plotted on the chart based on how many requirements and derived requirements the methodology was a candidate for meeting. The resulting design space mapping is shown in Figure 18 – Identified Candidate Methodologies Vs. Requirements.

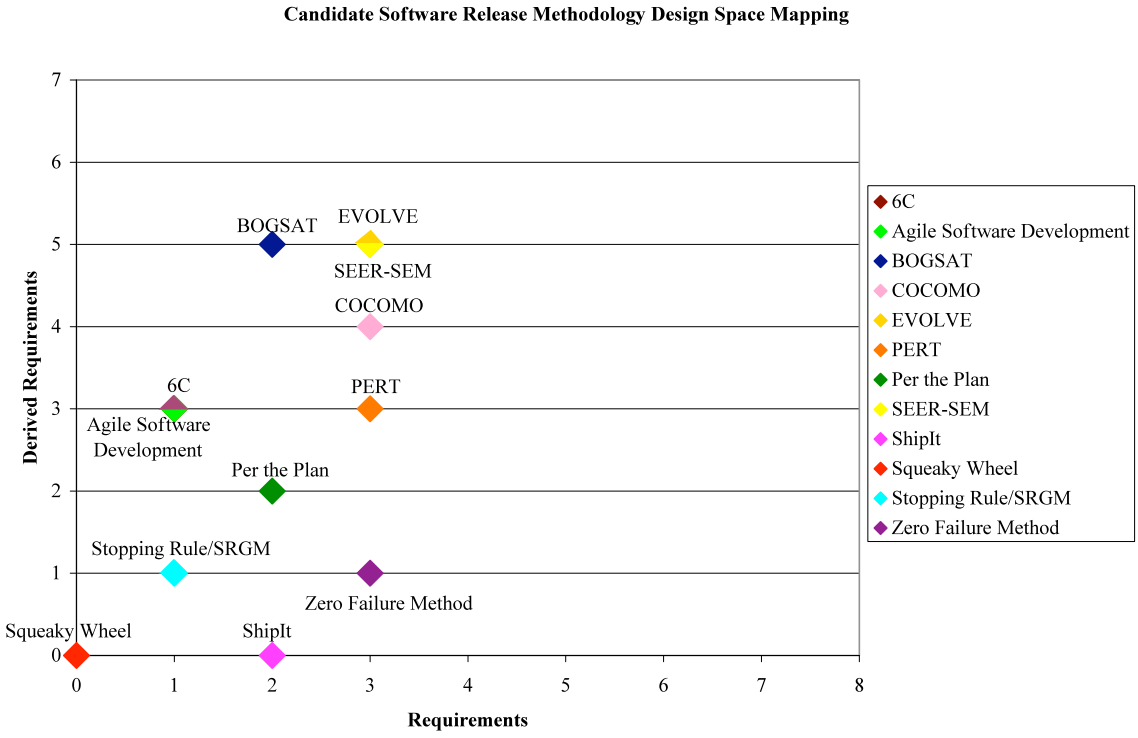

Figure 18 – Identified Candidate Methodologies Vs. Requirements

Using this type of design space mapping, the candidate methodologies that meet all the requirements would be shown plotted in the upper right corner and those that meet none of the requirements would be plotted in the lower left corner of the plot. The specific requirements that each methodology is a candidate for meeting, or not, are not shown. But the visual representation of the methodology's ability to meet a number of requirements is shown, providing the reviewer with the ability to rank the methodologies against one another with respect to their ability to meet a number of requirements.

Analysis of the design space mapping indicates none of the identified candidate methodologies are plotted in the upper right corner, indicating that no candidate

methodologies meet all the requirements and derived requirements for the mission critical release readiness methodology. None of the methodologies are plotted in the right half of the plot, indicating the methodologies did poorly against their ability to meet the requirements. Several of the methodologies are plotted in the upper left section, indicating those identified candidate methodologies did well with their ability to meet derived requirements (BOGSAT, EVOLVE, and SEER-SEM), but none of the identified candidate methodologies met all of the derived requirements.

The analysis of the design space mapping indicate there was no one identified candidate methodology that meets all the requirements and derived requirements for the mission critical release readiness methodology. At this point there are three paths that one could follow to develop an improved methodology for releasing mission critical software: 1) Perform additional research to determine if a software release methodology has been developed that would meet all or at least more of the requirements; 2) Attempt to combine two or more methodologies to meet more requirements; or 3) Develop a methodology that better meets the mission critical release readiness methodology requirements.

The decision as to which path to take appears to be clear after reviewing the options. Research performed to date has not been exhaustive, but the results are not promising for the amount of research performed to the number of requirements met by the researched methodologies. Combining methodologies does, at first glance, show some promise, but analyzing the set of requirements met by the methodologies taken as a whole, indicates several requirements were not met, meaning the requirements will not be met by any combination of methodologies. With the outcomes of the first two paths in

question after a quick review, the third path appears to be the logical path to follow – develop the methodology.

4.5 Methodology Development Requirements

Deciding to develop the mission critical readiness release methodology does not change the problem identified or the requirements for the methodology. However, new requirements may be required to cover the development of the mission critical release readiness methodology. Before researching whether the requirements require updating, the goals of the methodology will be discussed. Additional requirements analyses will be re-examined after the goal discussion.

4.5.1 Decision Analysis

Since the overarching goal of the mission critical release readiness methodology is to assist the software release manager in deciding when to release mission critical software, the methodology requires a basis in decision analysis. For the software release manager, after analyzing the software release decision and evaluating the possible outcomes, a decision must still be made [75].

The software release decision must take into account both quantitative and qualitative issues. Decision automation is only possible when quantitative analysis is used without qualitative issues, but in most situations, quantitative analysis will be used as an aid to make decisions [57]. Wasson would place the software release decision as analytical decision support practice [40].

4.5.1.1 Mission Critical Release Readiness Methodology As A Decision Tool

The mission critical release readiness methodology will be required to consider the incomplete state of the program that exists for an interim release and factors that could affect a software release such as cost, customer satisfaction, problem reports, resources, requirements, risk, software criticality, software security, and schedules. The mission critical release readiness methodology will assist system development programs in determining the optimal time to release an interim software release that supports its intended purpose, given multiple integrated software products, while considering the factors mentioned above.

The proposed mission critical release readiness methodology is not meant to replace software release planning, but aid in the software release decision process. The methodology will use the software plan as an input to the decision matrix to assist in determining the optimal release time for a specific interim software release. The mission critical release readiness methodology is not meant to solve the question of when to release the software, but to assist the decision makers in their decision making process of when to release software by incorporating the program's current state, software release process, current software plan, resource availability, software dependencies, problem reports, and requirements into an analytical factor. The methodology's benefits will be especially useful as the decision of when to release software becomes more difficult:

> More difficult decision problems are naturally more difficult to analyze. This is true regardless of the degree to which formal analysis (i.e., use of models as a decision aid) or intuitive appraisal (i.e., in one's head) is used. However, as

complexity increases, the efficacy of the intuitive appraisal decreases more rapidly than formal analysis [9].

Software release decisions are difficult by themselves, but when combined with the problems of multiple integrated software products, and the ever changing development environment, there may be too much information for the decision maker to process the software release decision analytically. The decision maker may then use simplified mental strategies, without using decision analysis methods [10]. The mission critical release readiness methodology will analyze the data provided and provide an analytical aide to assist in the decision making process, with the goal of replacing non-productive, subjective, decision methodologies currently in use, like BOGSAT.

4.5.2 Reducing The Number Of Software Releases

Ideally, the mission critical release readiness methodology's analytical basis will aid in the software release decision process by providing analytical recommendations for releasing software. By basing software releases on analytical factors, the software released will be better able to support its intended purpose, which should reduce the number of releases not able to support their intended purpose and therefore reduce the number of software releases.

A goal of the mission critical release readiness methodology is to reduce software releases. That's a good thing, right? If it is just software, why not release it anytime? While it is true that software can be released anytime, cost and schedule normally constrain the number of software releases for a particular program. Releasing software incurs schedule and monetary costs while reducing resource availability. It takes a finite

amount of time to make, build, release, document, and test a software release. During the release, the resources used (people, computers, labs, etc.) are not available to perform other tasks (incurring schedule costs and reducing the resource's availability) and personnel must be paid for their time (incurring monetary cost). Consequently, reducing the number of software releases performed due to poor software release decisions reduces the overall cost of the program.

4.6 Requirements Update

The goals of the mission critical release readiness methodology development are to assist in deciding to release software via an analytical basis and to reduce the number of software releases. The goals of developing the mission critical release readiness methodology are covered by the previously developed requirements and therefore no new requirements will be added, nor will any requirements require an update at this time. Ideally, the mission critical release readiness methodology should be able to meet all the requirements and therefore plot in the upper right corner of the design space mapping chart. A new design space mapping with the mission critical release readiness methodology will be completed in a later section, after the requirement verification to show where the developed methodology maps against the identified candidate methodologies.

Analyzing the design space mapping points out another reason to develop the mission critical release readiness methodology rather than researching or combining methodologies, because the requirements are unique to the problem under consideration, it is doubtful that any single or even combination of methodologies could have met all the

requirements. Knowing what the requirements are and being able to design the mission critical release readiness methodology to meet the problem specific requirements further indicates that developing a methodology is the most logical path.

4.7 Methodology Risk

The *Risk Management Guide For DoD Acquisition* defines risk as "a measure of future uncertainties in achieving program performance goals and objectives within defined cost, schedule and performance constraints" [65]. The five steps of risk management, as identified by the Department of Defense (DoD), are identification, analysis, mitigation planning, implementing risk mitigation plans, and risk tracking. The DoD risk process is shown in Figure 19. Risks for the mission critical release readiness methodology are identified, mitigations planned, and risks tracked via a modified DoD risk process.

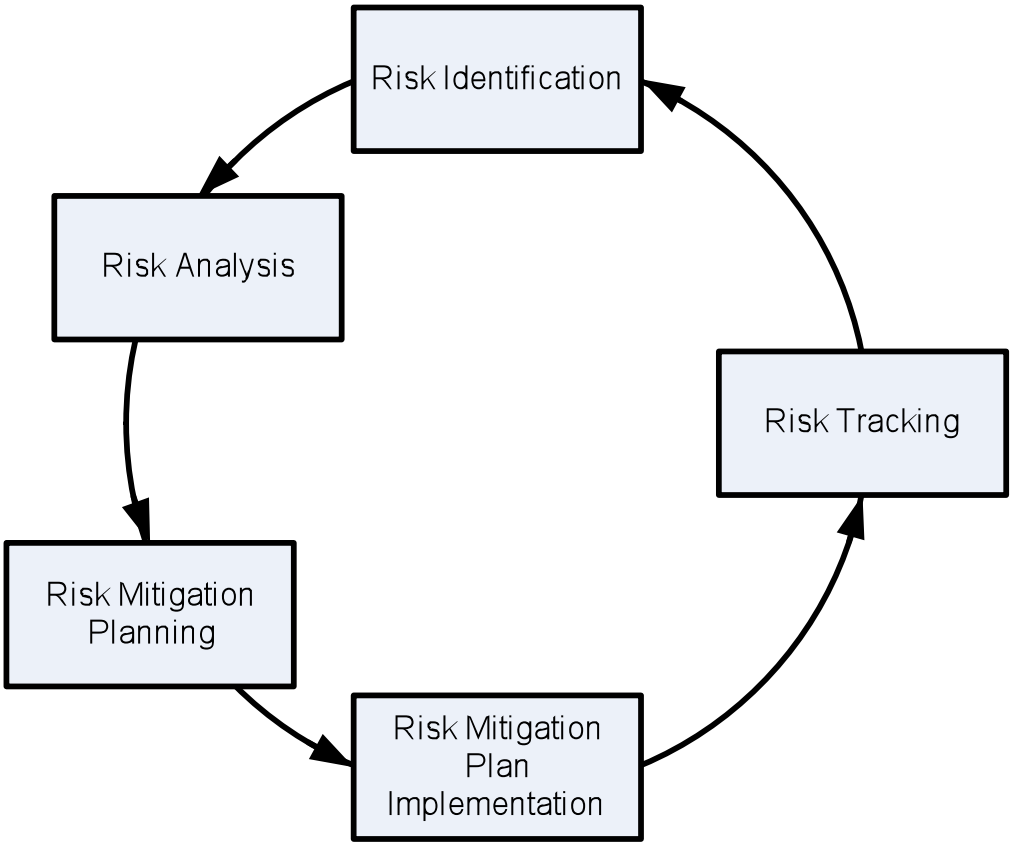

Figure 19 – DoD Risk Management Process

Risk tracking is accomplished by plotting the risks on a two axis, 5 x 5 matrix. The axes of the matrix are the risk consequence and the likelihood of the risk occurring. A sample risk matrix is shown in Figure 20. The consequence of the risk is shown in the "X" axis and the likelihood of the risk is shown in the "Y" axis. The matrix uses colors to identify low, medium, and high risks. Low risks are shown in green, medium in yellow, and high risks are shown in red.

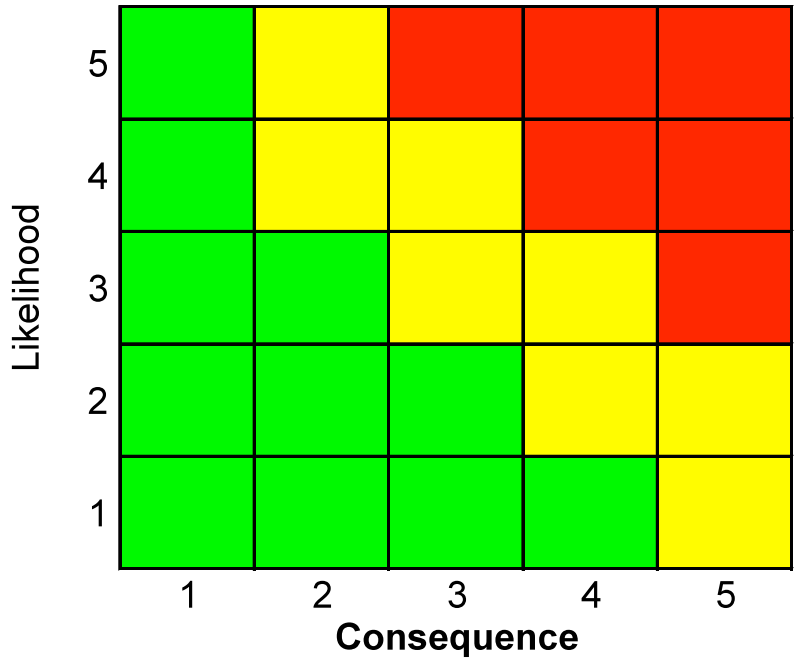

Figure 20 – Risk Matrix

The likelihood axis of the matrix is the likelihood or probability of a risk occurring. The likelihood levels and their probabilities, as developed for the software release decision methodology, are shown in Table 5.

Table 5 – Risk Likelihood

LIKELIHOOD					
Level	**1**	**2**	**3**	**4**	**5**
Probability	**0 to 15%**	**15 to 40%**	**40 to 60%**	**60 to 85%**	**85 to 100%**

Typically, the consequences of a risk occurring would be ranked across technical performance, schedule, and cost, but due to the nature of the mission critical release readiness methodology development, only technical performance is a valid risk consequence category. The risk consequence levels, as developed for the mission critical release readiness methodology, are shown in Table 6.

Table 6 – Risk Consequence

CONSEQUENCE					
Level	**1**	**2**	**3**	**4**	**5**
Consequence	Minimal or no impact on using methodology.	Affect on usage is tolerable, with little to no impact to timeliness.	Workarounds required to use methodology and/or methodology requires up to 2 hours to process.	Affect on methodology usage is almost intolerable and/or methodology processing greater than 2 hours.	Methodology is not usable.

Two risks were identified related to the software release decision methodology:

1. If the methodology is inefficient at gathering and processing the methodology inputs, then the effect on the methodology would be to use workarounds or add up to 2 additional hours to the processing. (Medium Risk)

2. If the methodology does not easily apply to a wide variety of software development methodologies and tools, then the methodology will not be widely acceptable and the methodology would be considered almost unusable. (High Risk)

The software release decision methodology identified risks are shown plotted on a risk matrix in Figure 21. The risks and their mitigation plans developed to aide in the reduction of the risks as the software release decision methodology is designed and implemented are shown in Table 7 – Methodology Risk Mitigations.

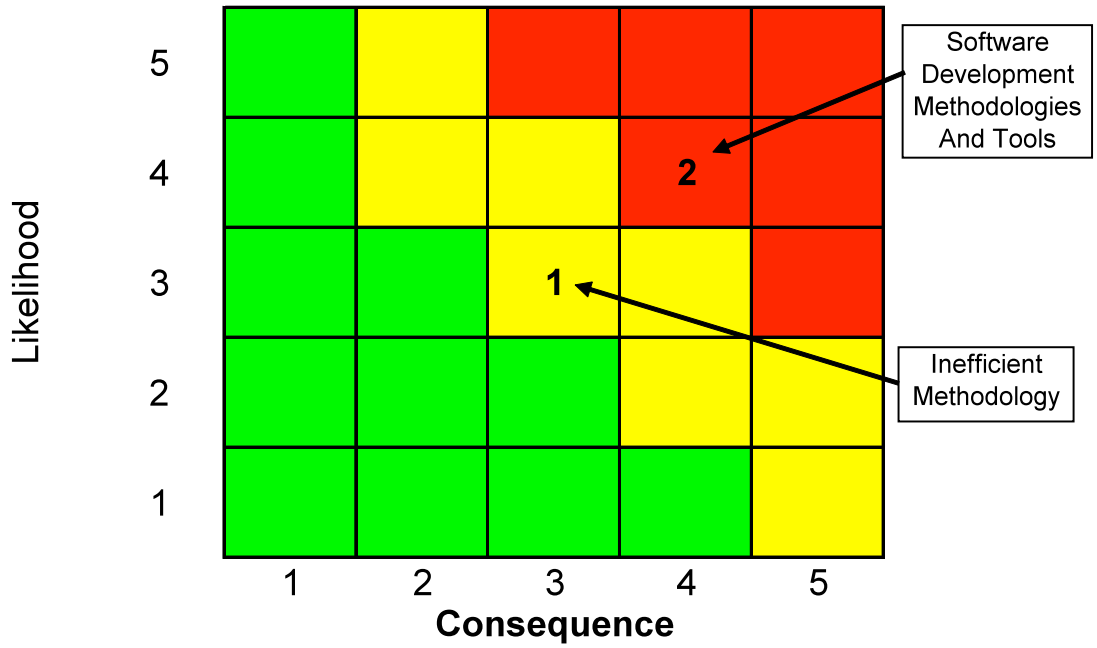

Figure 21 – Methodology Risks

Table 7 – Methodology Risk Mitigations

RISK	MITIGATION PLAN
1. If the methodology is inefficient at gathering and processing the methodology inputs, then the affect on the methodology would be to use workarounds or add up to 2 additional hours to the processing. (Medium Risk)	1. Design methodology to use common tools and interfaces. 2. Automate portions of methodology as time allows. 3. Provide standardize data entry forms.
2. If the methodology does not easily apply to a wide variety of software development methodologies and tools, then the methodology will not be widely acceptable and the methodology would be considered almost unusable. (High Risk)	1. Use generic software development labels on data entry forms. 2. Design in methodology customization, where possible.

4.8 MCRRM Design Space and Analysis Summary

Methodologies for releasing software were researched and compared against the requirements developed for the mission critical release readiness methodology. The design space mapping of the methodologies against the number of requirements and derived requirements that the identified candidate methodology might meet indicated none of the methodologies met all the requirements and the best methodologies only met just over half of the requirements. Further analysis indicated redundancies in requirements met by the individual methodologies. These redundancies in meeting requirements indicate that combining identified candidate methodologies for additional requirement coverage does not appear to be a feasible pursuit, nor did further research on

additional software release methodologies. Consequently, it was decided that developing a unique methodology would be required to meet the requirements of mission critical release readiness methodology.

Goals for developing the mission critical release readiness methodology were developed as a check for requirement coverage against existing requirements. The goals identified were analytical decision analysis – assist in the decision of when to release mission critical software – and reducing the number of software releases. A review of the goals against the requirements previously developed for the mission critical release readiness methodology did not uncover any needed new requirements or updates to existing requirements.

A risk management methodology for development of the mission critical release readiness methodology is introduced and two risks for developing the mission critical release readiness methodology are identified. The risks are ranked as medium and high risk. The risk mitigation plans are developed for both of the identified risks and will be used as the mission critical release readiness methodology is developed to reduce the risk levels.

Chapter 5

METHODOLOGY DESIGN

5.1 Functional Analysis/Allocation

The systems engineering fundamentals process' *Functional Analysis/Allocation* sub-process decomposes functions into lower-level functions, producing a description of the product and its performance, with its outputs used to optimize physical solutions [17].

5.2 Choose Solution

In the need identification and problem resolution process the *Choose Solution* sub-process identifies the best solution from the design space formulated in the formulate solution design space sub-process performed earlier. A variety of tools may be used to assist in choosing the solution (Design Space Mapping, Quality Function Deployment (QFD), Functional Flow Block Diagrams (FFBD), AHP (Importance), TRIZ, Design Selection, etc.) The best solution may be found by using any one or combination of the tools mentioned.

Once the problem solution method has been chosen, the solution's inputs and outputs are identified. Identifying the Inputs/Outputs (I/O) form the basis for the solution development carried out in later sections.

5.3 Analyzing the Design Space

Analyzing the final design space mapping from chapter 4 indicates those methodologies with multiple inputs, SEER-SEM, EVOLVE, COCOMO, and even BOGSAT, met more requirements than those methodologies with fewer inputs. This indicates that the MCRRM will require multiple inputs to meet the requirements and the inputs must be considered during development of the methodology. Reviewing the derived requirements confirms this result, as all seven derived requirements define inputs to the methodology.

5.4 More ARSD Functional Flow Block Diagramming

With the research accomplished on software release methodologies and the software release manager's roles, the sub-function analyze software release decision for the top level software development process is developed into its sub-functions. Figure 22 – Analyze Software Release Decision shows the top-level software development process's functional flow block diagram with the analyze software release decision sub-functions, which are: 5.1.1 – Provide Inputs; 5.1.2 – Verify Software Release Process Followed; 5.1.3 – Analyze Release Against Requirements; 5.1.4 – Calculate Release Readiness; 5.1.5 Output Release Analysis.

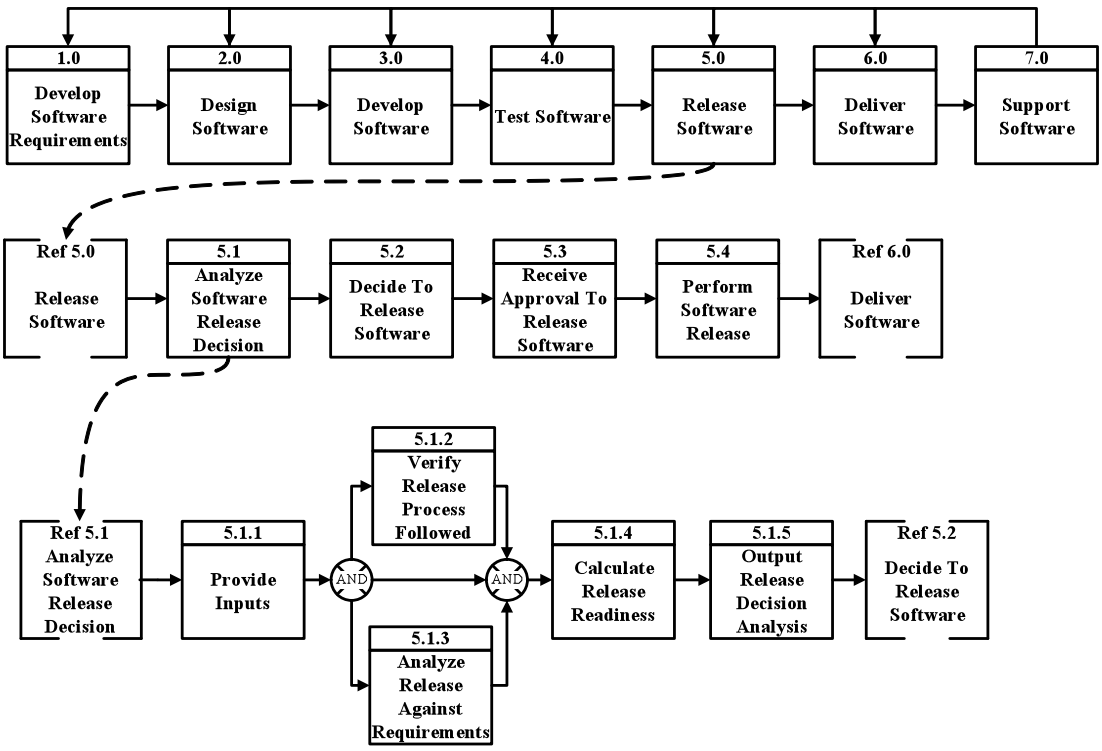

Figure 22 – Analyze Software Release Decision

As shown in Figure 22, the analyze software release decision sub-function includes two and-summing blocks. The first and-summing block is directly out of sub-function provide inputs and indicates that the inputs go to all three of the next sub-functions and that all three sub-functions must be completed before continuing in the function flow. The next and-summing block is anding the outputs of verify software release process followed, analyze release against requirements, and provide inputs. This and-summing block indicates all three of the previous sub-functions (provide inputs, verify software release process followed, and analyze release against requirements) outputs are required before completing the calculate release readiness sub-function.

The analyze software release decision sub-functions are discussed in more detail in the following chapter, during the design of the mission critical release readiness methodology however, top-level overviews of the sub-sections and their functions are provided in the following sections.

5.4.1 Provide Inputs

The analyze software release decision sub-function provide inputs needed to perform the verification, analysis, and calculations of the analyze software release decision sub-functions.

5.4.2 Verify Software Release Process Followed

As mentioned earlier, following a process for software development and release is a key step to generating reliable and dependable mission critical software. The verify software release process followed sub-function provides the software release manager the verification that the software release process is being followed for the software release under consideration.

5.4.3 Analyze Release Against Requirements

The sub-function analyze release against requirements, analyzes the software release against the requirements for that software release to insure the software is able to perform its intended function.

5.4.4 Calculate Release Readiness

The sub-function calculate release readiness is where the decision to release or not is calculated. The sub-function uses the provided inputs to produce a release readiness that assists the software release manager in the decision to release software or not. This sub-function and the output release decision analysis sub-function are key to the mission critical release readiness methodology.

5.4.5 Output Release Decision Analysis

The previous sub-function calculated the software release readiness, the output release decision analysis sub-function presents the calculated release readiness for the software release manager's use in deciding whether to release the software or not.

5.5 Input/Output (I/O) System Diagram

In the analyze software release decision sub-function, two sub-functions, provide inputs and output release decision analysis, are associated with the input and output of the function. An Input/Output (I/O) System Diagram of the analyze software release decision sub-function is developed to understand the function and to assist in the design and development of the MCRRM in later chapters.

The I/O Diagram for the analyze software release decision sub-function was developed by; analyzing the requirements developed in section 4 for inputs and/or outputs; reviewing the software release, software, and program manager's job duties for any duties that required inputs and/or outputs; and analyzing the Analyze Software Release Decision FFBD, Figure 22, for input and/or output requirements. The research

into the function inputs discovered seven inputs, all of which could be considered normal program data for a program developing mission critical software.

The analyze software release decision sub-function output research discovered five outputs for the function. Four of the five function outputs were to simply answer questions. The questions are related to additional functions required to be performed or affect the ability to release the software itself. The I/O System Diagram developed is shown in Figure 23. The inputs and outputs shown in the diagram are discussed in greater detail in the following sections.

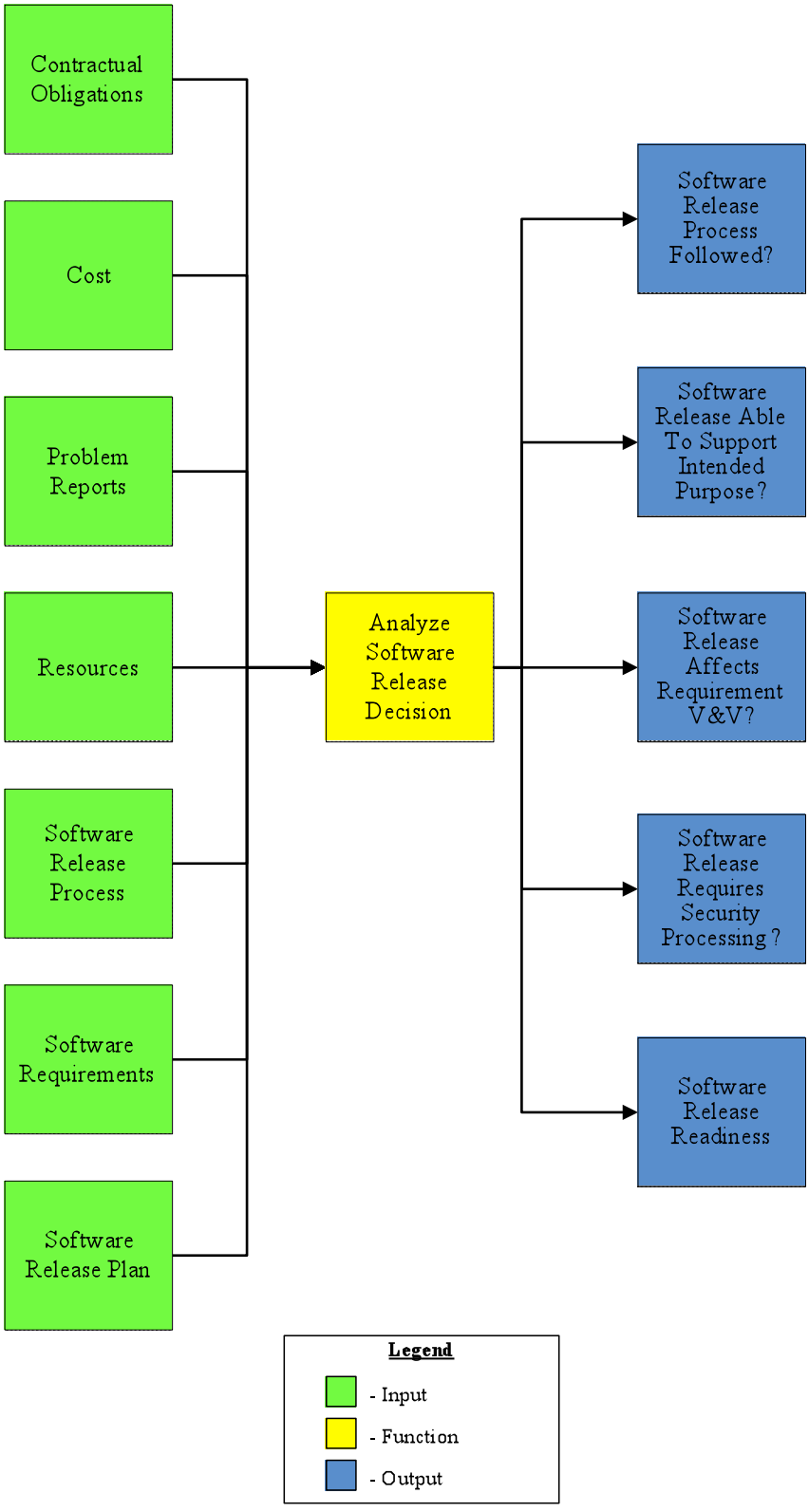

Figure 23 – Analyze Software Release Decision I/O System Diagram

5.6 Analyze Software Release Decision Inputs

The analyze software release decision sub-function requires timely, accurate, and relevant input data as the function calculates the software's release readiness to assist the software release manager in deciding to release software. In order to assist in the release decision making process, the analyze software release decision sub-function receives inputs from various other functions and/or activities, (problem reporting, test, program management, etc.). These inputs are currently available to the software release manager, but not in a format that can be used for analytical decision making purposes. The analyze software release decision sub-function receives the inputs, analyzes them and uses them to provide the required outputs including an analytically based software release readiness output, all in a format the software release manager can use to assist in deciding when to release software.

The analyze software release decision sub-function does not require the creation of data for inputs. It uses existing program data as its inputs. The analyze software release decision sub-function inputs are shown below:

1. Contractual Obligations
2. Cost
3. Problem Reports
4. Resources
5. Software Release Process
6. Software Requirements
7. Software Release Plan

The function inputs are briefly described in the following sections. Additional details will be provided in later sections as needed.

5.6.1 Contractual Obligations

The contractual obligations input to the analyze software release decision sub-function includes all contractual items that are affected by the software under consideration for release. Contractual obligations may be requirements flowed down from the contract to the software under development or perhaps a milestone in the project schedule. Requirements are also input to the analyze software release decision sub-function, but the contractual nature of these requirements requires extra scrutiny by the software release manager. For instance, a contractual obligation may require customer participation/review of the obligation and non-contractual obligations may permit the customer to participate/review the obligation, but customer participation is not required. The contractual obligations input will be a listing of obligations that will be supported by the software under consideration for release and their dates.

5.6.2 Cost

Costs vary not only from project to project, but from industry to industry. A multi-thousand line software development project's cost will be very large compared to a one line software change to fix a bug. However, assuming the two releases are from the same project, the release costs for the multi-thousand line software development and the bug fix are probably the same and the bug release's development costs may be equaled

by the cost of releasing the software [64]. As software projects grow in size and complexity, so do the costs to release software. Additionally, the criticality of the software being released affects the cost of the release. The space shuttle's software release cost will be very different from a hobbyist's release cost for a simple iPhone game.

Software release costs could be calculated by adding up the time a person spent gathering the software, compiling the software, and filling out the release documentation. This release cost calculation may work fine for a small, less complex project.

However, the time spent gathering, compiling, and documenting the release is a small part of the release cost for a complex system. Besides releasing the software, the release must be tested. Software may go through several tests: unit, standalone, integrated, system, and depending upon the software's intended purpose: ground, flight, acceptance, and/or security. The release test duration for a complex system can be anywhere from several hours of testing to several weeks. Additionally the release must be documented for the maintainers and the operators of the system and the support and operator's manuals documentation may take several weeks to complete, adding to the release cost and schedule. For the mission critical release readiness methodology, it is suggested that the cost be a roll up of man hour costs for testing, documenting and releasing the software. Additional project specific cost data can be added to the recommend cost data, as required.

5.6.3 Problem Reports

When a problem is found during the software development process, the problem is written up and tracked. The problem discovered may go by many names: bug; software problem report; software problem anomaly report; problem report; defect; etc. The identified problems are tracked in a problem reporting system that tracks the description of the problem and key information regarding the problem and its status in the system.

The analyze software release decision sub-function's problem reports input is derived directly from the problem reporting system. The problem reports input may vary by project depending upon the project's criticality, safety requirements, and problem reporting process. The number and severity of problem reports may affect the software release decision. For instance: high severity problem reports for the space shuttle software do not allow testing to begin [42].

Further details on the problem reports inputs will be provided during the development of the methodology in a later section, but recommended minimums for the analyze software release decision problem reports input would be a listing of all open and closed problem reports including problem report status, subsystem/system affected, software versions affected by the problem, problem report severity, subsystem/system functions affected, planned software release implementation, and any mitigations in place. Due to their affects on software's ability to perform its intended function, problem report severity and mitigations are discussed in the next sections.

5.6.3.1 Problem Report Severity

Problem report severity is an attribute of a problem report and is provided as an input to the analyze software release decision sub-function via the problem reporting process. The American Heritage® Dictionary of the English Language defines severity as "The state or quality of being severe." Typically the severity of a problem report is broken into a fixed number of states of varying consequences describing the problem reports effect on the system or program. The exact number of the states and their definitions is somewhat of a subjective science and various methods are used to determine them [7]. There is a fine line between too many states and too few, previous experiences have shown that somewhere between 4 and 6 states are ideal. Given 5 states, the consequence of the severity may be applied in to different aspects of the program in a manner similar to how NASA applies criticality categories [3]. NASA has three categories, with each of the first two categories having subcategories for redundant hardware elements.

By building upon the NASA usage and amending the categories, an example of problem report severity states and consequence definitions is developed for use in the analyze software release decision sub-function. Five severities are identified, 1 through 5, with 1 being the most critical severity. Additionally, severities 1 and 2 have two subcategories Intended Purpose (I) and Safety (S). The developed problem reporting severity states developed are shown in Table 8 – Example Problem Reporting Severity States.

Table 8 – Example Problem Reporting Severity States

Severity	SubCategory	Consequence
1	I	Software is not able to complete intended purpose.
	S	Possible loss of life.
2	I	Intended purpose is severely affected.
	S	Injury to personnel
3		Intended purpose affected, but a work-around is known.
4		Nuisance affects but intended purpose is still supported.
5		None of the above.

To properly classify the severity of a problem report of severity 1 or 2, the subcategory must accompany the severity number.

5.6.3.2 Mitigate Problem Reports

If a high severity problem report, say a one or two from the examples in the previous section, is assigned to software about to be released, the software may be unable to support its intended purpose and therefore not be releasable. Conversely, a high severity problem report written against software does not mean the software can not be released. The project may have mitigations in place that would negate the implications of a high severity problem report. For instance, that function may be disabled in the software release under consideration or words may be added to the operator's manual that prohibited the use of the system in a manner which may cause the problem report to repeat. If mitigations are in place, the severity of the problem report may be reduced to allow the software to be released. The checking of problem reports for mitigatibility should be incorporated into the normal project problem reporting process and not incorporated as part of the analyze software release decision sub-function. The

understanding of mitigatability is an important aspect of the software release manager's job and the software release decision process.

5.6.4 Requirements

The requirements input to the analyze software release decision sub-function are the applicable requirements for the current software under consideration for release. The analyze software release decision sub-function analyzes the applicable requirements against the problem reports against the software to output whether the software release under consideration can support its intended purpose and whether requirement verification and validation are impacted by the software release. The actual data input to the sub-function are requirement wording, requirement status, subsystem(s)/function(s) implementing the requirement, and planned requirement verification and validation dates.

5.6.5 Resources

The analyze software release decision resources inputs are those resources that affect the software release decision. The resources affecting the software release decision may vary by project size and complexity, but in general will include, but not be limited to: personnel; testing laboratories; test articles; test facilities; hardware; release software; test software; support software.

5.6.6 Software Release Plan

The software release plan input to the analyze software release decision sub-function is the project plan for releasing software. The software release plan will include

the scheduled release date for the software under release consideration and allow the mission critical release readiness methodology to analyze the planned schedule against current events and the projected scheduled release of software.

5.6.7 Software Release Process

It is critical for the software release manager to insure the software release process is followed when releasing software. For this reason, the software release process is an input to the analyze software release decision sub-function to insure the software release plan followed the release process and to insure the software release manager and project management know when processes are being followed and more importantly, when processes are not being followed.

5.7 Analyze Software Release Decision Sub-Function Outputs

The analyze software release decision sub-function receives the inputs discussed above, analyzes them, and calculates outputs to assist the software release manager in deciding when to release the software. The analyze software release decision sub-function outputs are derived from an analytical basis and not only assist in the decision making process but also allow the software release manager to present an analytical basis for the software release decision to management. There are five analyze software release decision sub-function outputs and they are listed below:

1. Software Release Process Followed?
2. Software Able To Support Intended Purpose?

3. Software Release Affects Requirement V&V?
4. Software Release Requires Security Processing?
5. Software Release Readiness

The analyze software release decision sub-function outputs are briefly described in the following sections with additional details provided in later sections, as needed.

5.7.1 Software Release Process Followed?

The knowledge of whether a software release follows the software release process is important when dealing with critical software, be it safety or mission critical software. The analyze software release decision sub-function will provide an output that indicates whether the release process was followed or not. Just because a release option does not follow the release process does not mean the software release manager will choose not to release the software.

If the software release process was not followed, it would be an indication that the release process may require modification. Not all software release decisions will be yes or no; some may to work in the grey area of release processes.

For instance:

- Given a software release process that requires release testing be complete before releasing the software;
- The customer is in to validate the system displays – which have been thoroughly tested – for one day only;

- Release testing is not complete;
- Then the software could be released to support customer validation of the system displays and the release testing could be identified as complete for the intended purpose, system displays validation.

In the example presented, there is a compelling reason to release software without completing full release testing, but by declaring the release testing to be the subset of testing that supports the software's intended purpose, the software can be released within the existing software release process, albeit by operating in a process grey area. Another solution, instead of operating in a process grey area, would be to modify the software release process to allow modification of the set of release tests required before release. Processes are open to change and even when dealing with critical software, the software release process can be amended to allow a release option to follow a modified software release process.

5.7.2 Software Able To Support Intended Purpose?

A key factor incorporated into any decision on releasing software is "can the software support its intended purpose?" The more complex a system is, the harder it becomes to determine if the software can support its intended purpose. Depending on the system, the software's intended purpose may change from release to release. One software release may be intended as a test release to test the software build process, the next release as a lab release to test basic functions, the next release a ground test only release for integrating the software on the ground, then, depending on the system, maybe

a flight test release for flight testing, and then, finally, a production software release for delivery to the customer, but maybe with limited functionality – meaning yet more releases. The mission critical release readiness methodology will analyze the provide inputs and calculate a ranking against the software's intended purpose to assist the software release manager in deciding when to release software.

5.7.3 Software Release Affects Requirement V&V?

Although Verification and Validation (V&V) could be considered an intended purpose for the software, they are such key activities for most projects with wide ranging implications; affecting V&V is broken out as a separate analyze software release decision output. If a software release affects the ability to perform V&V the software release manager will be notified and able to factor that information in the software release decision making process.

5.7.4 Software Release Requires Security Processing?

Software security is a major concern in today's software environment. If a software release requires security processing, additional activities are required to insure the software is secure. The analyze software release decision sub-function outputs whether the software requires additional security processing to insure the software release process is followed and that the software release manager understands additional processing is required before releasing the software. The security processing required will vary from project to project and for the software's intended purpose. A lab release

requires may not require security processing, whereas a production release to the customer will require the extra processing.

5.7.5 Software Release Readiness

The analyze software release function calculates the software's release readiness and outputs it to provide an analytical basis to assist the software release manager in deciding when to release software. Given all the data regarding releasing software and multiple outputs from the analyze software release decision sub-function; having an analytical based ranking of the software release readiness will assist in the software release decision making process. The software release readiness output is the key output in the analyze software release function and the mission critical release readiness methodology. The development of how the software release readiness output is calculated and presented to the software release manager, while providing a managerial role-up of the methodology results are shown in the next chapter.

5.8 I/O Functional Flow Block Diagrams

The inputs and outputs for the analyze software release decision sub-function identified in the previous sections are added to the top-level software release process functional flow block diagram in Figure 24.

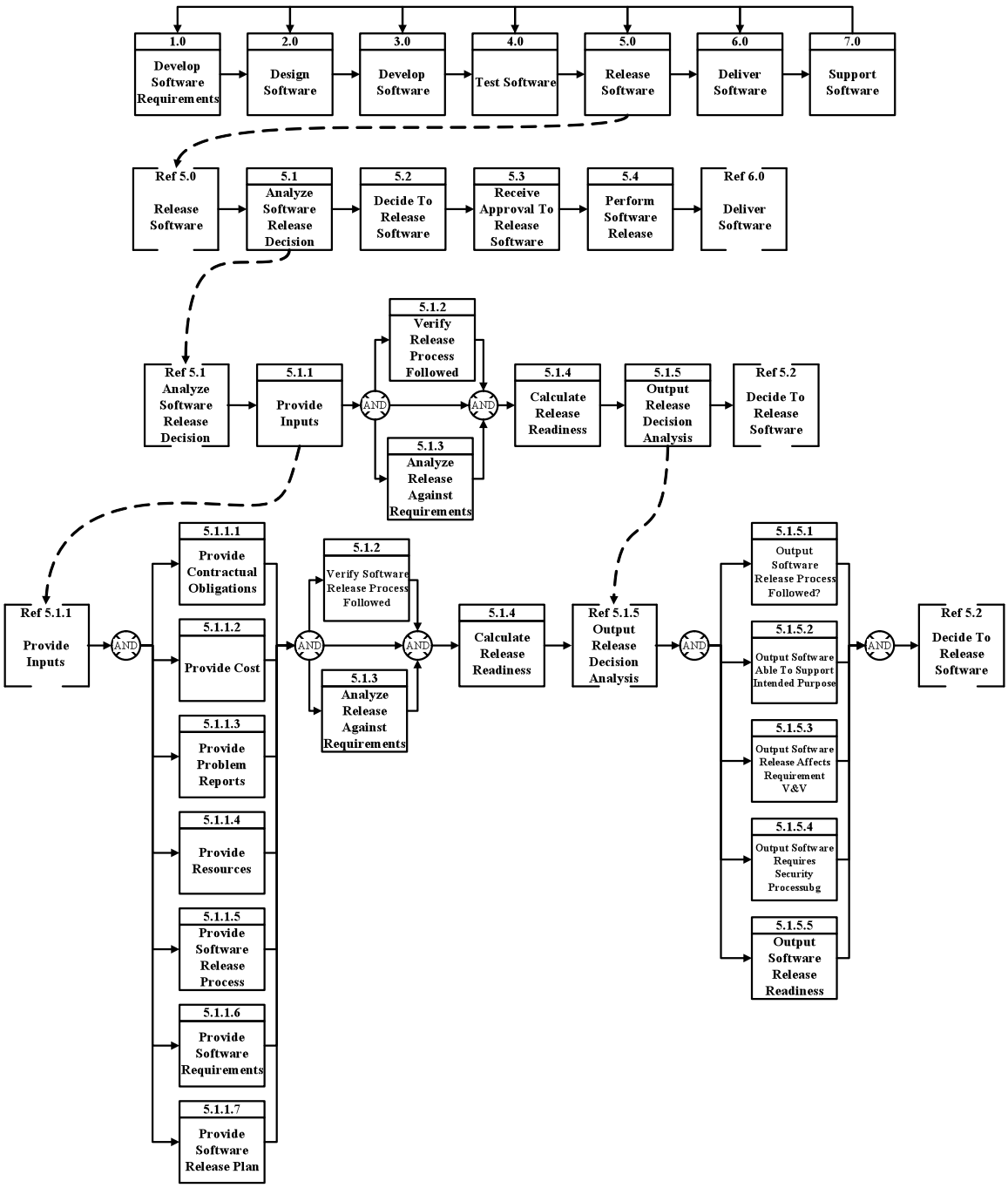

Figure 24 – I/O FFBD

The input/output portion of the analyze software release decision sub-function is shown in Figure 25.

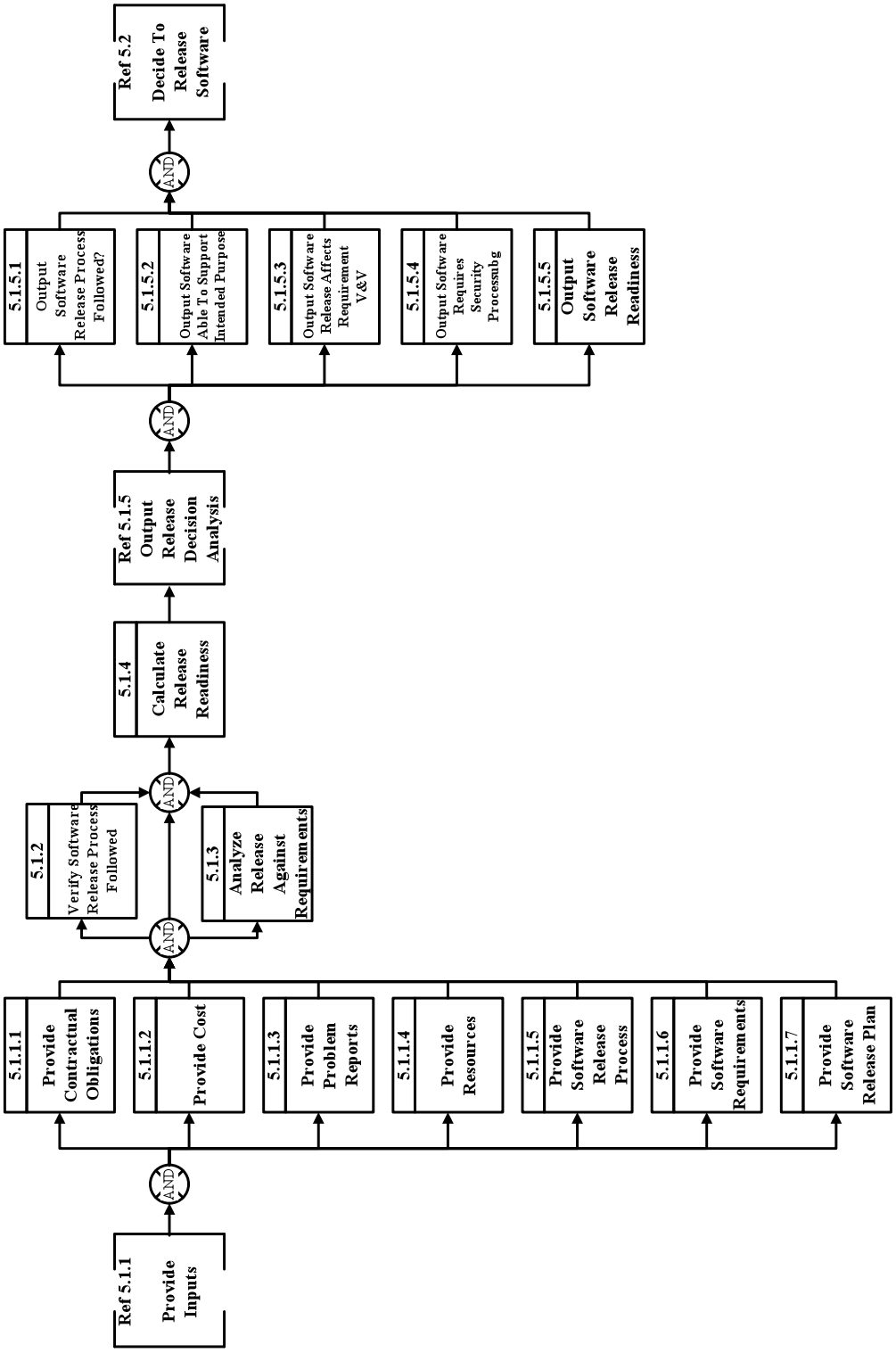

Figure 25 – Analyze Software Release Decision

5.9 Methodology Design Summary

The design space mapping performed earlier is the basis for additional methodology developmental activities. A function flow block diagram and I/O diagram were constructed to assist in the development of the analyze software release decision sub-function, the key function for the mission critical release readiness methodology. The functional flow block diagram identified several sub-functions of the analyze software release decision sub-function. The I/O diagram identified the inputs and outputs for the analyze software release decision sub-function while providing a graphical representation of the inputs and outputs for use during the development of the mission critical release readiness methodology.

The analyze software release decision inputs and outputs are discussed in generic, basic terms, over several sections. Discussing even the basics behind the I/O forms factual representations of how the data will be used by the analyze software release function and the mission critical release readiness methodology and why the inputs and outputs were chosen. The identified inputs and outputs are used to develop a final software development process functional flow block diagram. The release software function's, the analyze software release decision sub-function's, the provide inputs sub-function's, and the output software release readiness sub-function's functional flow block diagrams are shown on the software development process functional flow block diagram.

The design space mapping, function flow block diagrams, input/output diagram and discussions on the inputs and outputs are required for the development of the mission critical release readiness methodology, detailed in the next chapter.

Chapter 6

DEVELOPING THE METHODOLOGY

6.1 Design Synthesis

The systems engineering fundamental process' *Design Synthesis* sub-process is focused on the development of the physical architecture [17]. Typically developing the physical architecture is a combination of defining the hardware and software for the system under consideration. In this particular case, the development of a methodology, there is no hardware involved. For this case the *Design Synthesis* sub-process will focus on the development of the process behind the methodology and the analytical methods that support the methodology.

6.2 Develop Solution

The *Develop Solution* sub-process in the need identification and problem resolution process develops the solution selected under the Choose Solution sub-process. Upon completion of the *Develop Solution* sub-process, the selected solution shall be sufficiently developed and quantified to allow the testing of the solution to begin.

6.3 Development

Development of the mission critical release readiness methodology will be described in this chapter. The user needs, methodology requirements, methodology derived requirements, methodology development risks, process functional flow block diagrams, input/output diagram, and the basic analyze software release decision inputs and outputs have already been developed in previous chapters. In this chapter the previously accomplished research, analysis, and design are used to develop the mission critical release readiness methodology.

As described earlier, the mission critical release readiness methodology develops the processes and analytic methodologies for implementing the analyze software release decision sub-function in the release software function from the software development process. As the software development process was analyzed in the previous chapters, the inputs and outputs of the analyze software release decision were researched and developed. The analyze software release decision inputs and outputs are the basis for the mission critical release readiness methodology and its development and will be reviewed, as necessary, before developing the methodology.

As the mission critical release readiness methodology is developed: the analytical methodologies for calculating the outputs will be developed; a user interface for the methodology inputs will be developed; the final presentation of the mission critical release readiness methodology will be developed; as development proceeds, areas for future work will be indentified.

6.4 Inputs and Outputs Review

The inputs of the analyze software release decision sub-function are described generically as contractual obligations, cost, problem reports, resources, software release process, software requirements, and software release plan. The generic descriptors included some suggested data inclusion items regarding the inputs, but still generic basic software development project items. These generic descriptions follow the risk mitigation plan for the software development methodologies and tools risk item. The software development methodologies and tools risk item, with its mitigation plan is shown in Table 9. The use of the generic terminology allows the program using the mission critical release readiness methodology to apply their program specific input data and increases the likelihood of broader application of the methodology. Where possible, the development of the mission critical release readiness methodology will be developed with generic program data, allowing the programs to specify their inputs as the methodology is applied to their program. A generic software release process is presented in the next section as the software release process input to the mission critical release readiness methodology.

Table 9 – Software Mehodologies And Tools Risk Item

RISK	MITIGATION PLAN
1. If the methodology does not easily apply to a wide variety of software development methodologies and tools, then the methodology will not be widely acceptable and the methodology would be considered almost unusable. (High Risk)	1. Use generic software development labels on data entry forms. 2. Design in methodology customization, where possible.

6.4.1 Generic Software Release Process

The software release process is a required input to the methodology, based on derived requirement DR6 – the methodology shall use the current software release process as an input. Figure 26 – Generic Software Release ProcessFigure 26 is a generic software release process that will be used during the proofing of the mission critical release readiness methodology. It is expected that a program would input their unique software release process when using the methodology, but a software release process is needed to continue developing the mission critical release readiness methodology.

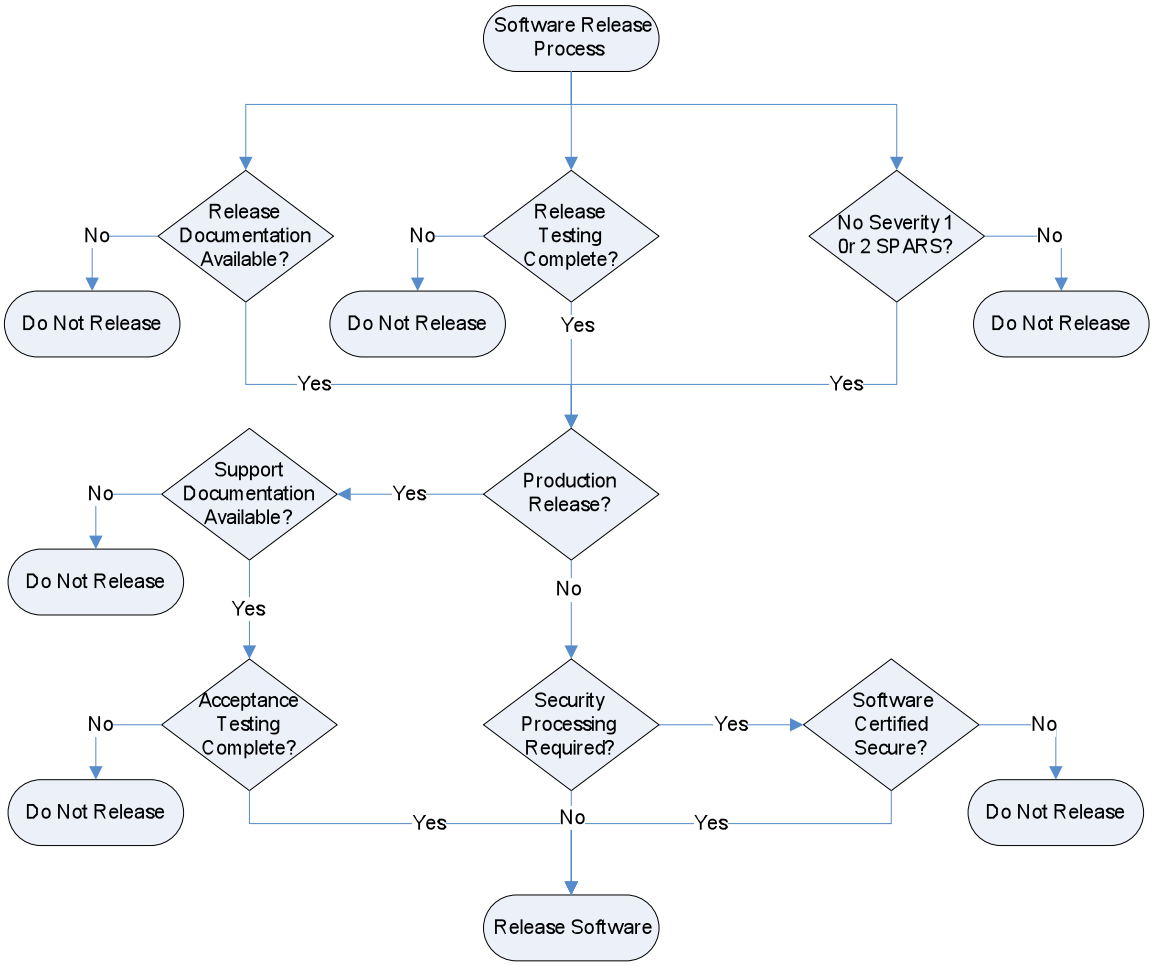

Figure 26 – Generic Software Release Process

The generic software release process identifies additional processing and testing required dependant upon whether the software release is a production release or requires security processing. The release process does not permit releasing software with out release document, with out release testing being complete or with severity 1 or 2 problem reports.

6.4.2 Outputs

The analyze software release decision sub-function outputs align well with the mission critical release readiness methodology requirements and will be the starting basis for the methodology's outputs.

6.5 Sub-Functions

This section will describe the development of the analytical methodologies for the release readiness metric as part of the mission critical release readiness methodology. In the analyze software release decision sub-function, shown in Figure 27 for reference, there are three sub-functions: 5.1.2 – Verify Software Release Process Followed; 5.1.3 Analyze Release Against Requirements; and 5.1.4 – Calculate Release Readiness. These sub-functions will be developed in the following sections.

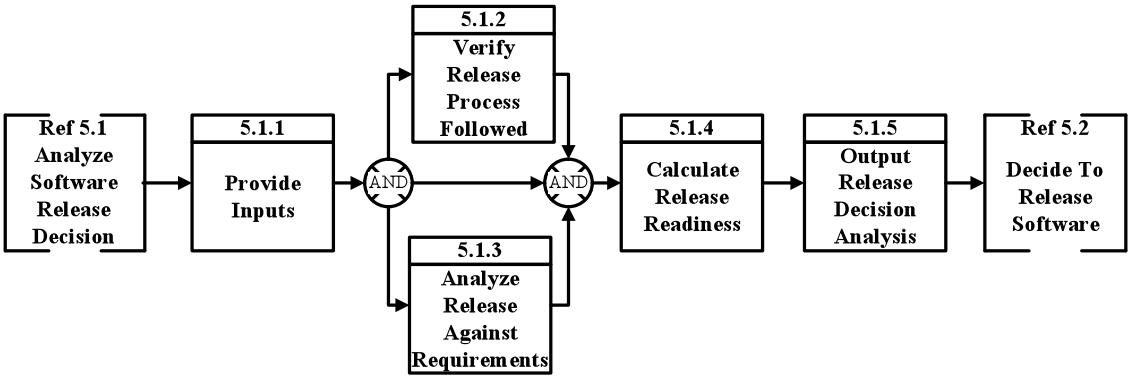

Figure 27 – ARSD FFBD

6.5.1 Verify Software Release Process Followed

The verify software release process followed sub-function answers the question whether or not the software release process was followed. The sub-function uses the software release process as its input and outputs whether the process was followed or not. The verify software release process followed functional flow block diagram is shown in Figure 28.

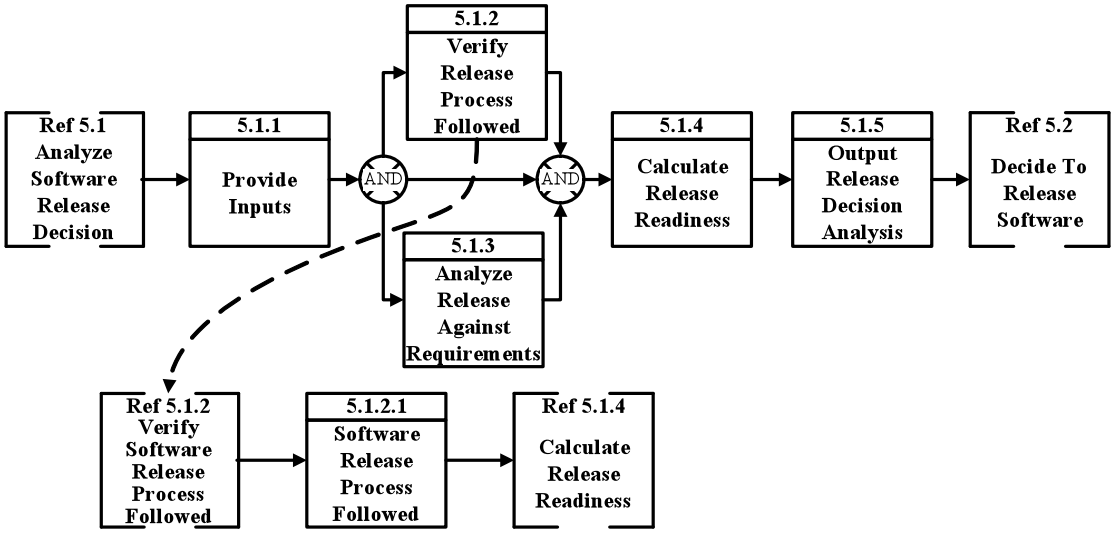

Figure 28 – Verify Software Release Process Followed

The verify software release process followed sub-function's methodology will be a simple answering of the question, "Was the software release process followed?" The answer will be input to the calculate release readiness sub-function and then presented to the user in the output release decision analysis sub-function.

6.5.2 Analyze Software Release Requirements

The analyze release against requirements sub-function uses the applicable software requirements and the problem reports as inputs and outputs whether any requirements verification and validation may be affected by the software release. This analysis assists the customer by giving providing quantifiable data regarding the verification and validation activities and whether any modifications to the activity will be required. The quantifying of the software's ability to meet the verification and validation activities also will be a factor in the ability of the software to meet its intended purpose. The analyze release against requirements function flow block diagram is shown in Figure 29.

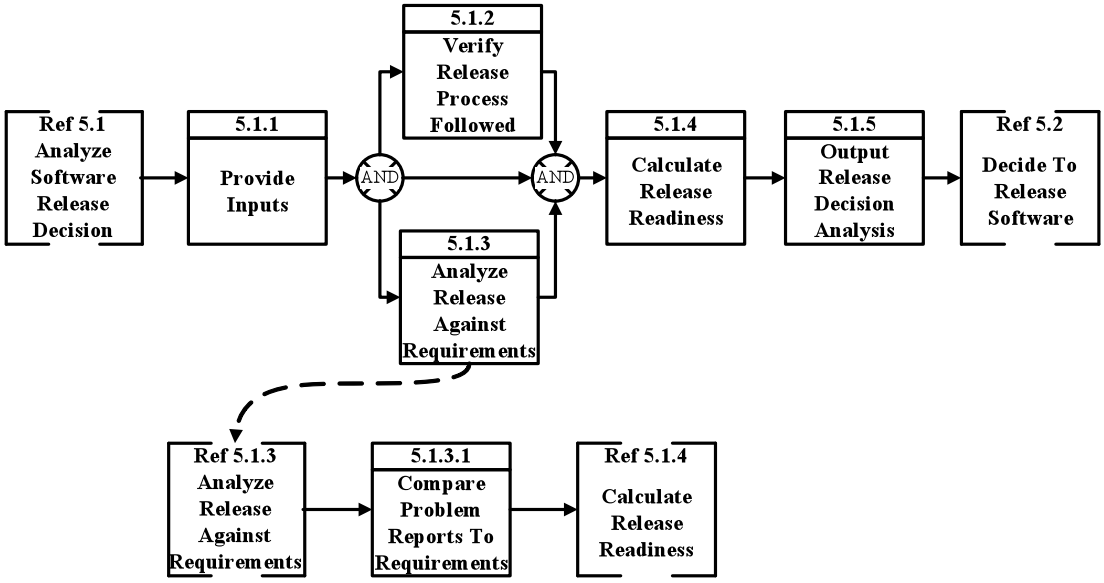

Figure 29 – Analyze Release Against Requirements

The analyze release against requirements sub-function's analyzes the open, applicable problem reports and the functions they affect, against the requirements and the

functions they implement. The sub-function factors the problem report's severity and the level of verification and validation of the requirement in determining whether the software release affects verification and validation activities.

6.5.3 Calculate Release Readiness Metric

As was stated in during the research, a metric based approach has considerable merit with regards to assisting the software release manager decide when to release software. As a decision problem with multiple decision criteria, the software release decision is classified as a Multi-Attribute Decision Making (MADM) problem. A method for solving multi-attribute decision making problems is the methodology proposed by Hwang and Yoon, the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) [76]. The TOPSIS method calculates one number that is an indication of an alternative's distance away from the ideal and negative-ideal solutions. The TOPSIS calculated number forms the basis for the mission critical release readiness metric. The TOPSIS technique is described below [72]:

Given m alternatives, A, and n criteria to rank the alternatives against perform the following:

1. Construct a decision matrix with the alternatives and criteria. The matrix will be described as follows:

$$D = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{22} & x_{22} & x_{23} & \dots & x_{2n} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix}$$

Where, m alternatives, i, are denoted as i = 1, 2, 3, …, m; the n criterion, j, are denoted as j = 1, 2, 3, …, n; and x_{ij} is the performance rating of each alternative with respect to each criterion.

2. Calculate a normalized decision matrix, R(=[r_{ij}]). The normalized values r, are non-dimensional criteria, converted from the performance rating, x, as follows:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}$$

Where, i = 1, 2, 3, …, m; j = 1, 2, 3, …, n. The normalized decision matrix, R, is described as follows:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1n} \\ r_{22} & r_{22} & r_{23} & \dots & r_{2n} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ r_{m1} & r_{m2} & r_{m3} & \dots & r_{mn} \end{bmatrix}$$

3. Calculate the weighted normalized decision matrix, V(=[v_{ij}]). The weighted normalized matrix is calculated by multiplying the normalized decision matrix by a set of weights W = (w_1, w_2, w_3, …, w_n) and ($\Sigma w_j = 1$), where j = 1, 2, 3, …, n and w_j is the weight of the j^{th} criterion. The weighted normalized value, v_{ij}, is calculated as:

$$v_{ij} = w_j r_{ij}$$

Where, i = 1, 2, 3, …, m; j = 1, 2, 3, …, n. The weighted normalized decision matrix, V, is described as follows:

$$V = \begin{bmatrix} w_1 r_{11} & w_2 r_{12} & w_3 r_{13} & \dots & w_n r_{1n} \\ w_1 r_{22} & w_2 r_{22} & w_3 r_{23} & \dots & w_n r_{2n} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ w_1 r_{m1} & w_2 r_{m2} & w_3 r_{m3} & \dots & w_n r_{mn} \end{bmatrix}$$

4. Calculate the ideal, A^*, and negative-ideal, A^-, alternatives as:

$A^* = \max(v_{ij})$ for positive criteria; $\min(v_{ij})$ for negative criteria for j = 1, 2, 3, …, n

$$A^* = [v_{1*}, v_{2*}, v_{3*}, \ldots, v_{n*}]$$

$A^- = \min(v_{ij})$ for positive criteria; $\max(v_{ij})$ for negative criteria for j = 1, 2, 3, …, n

$$A^- = [v_{1-}, v_{2-}, v_{3-}, \ldots, v_{n-}]$$

5. Calculate the separation measure using n-dimensional Euclidean distance method of each alternative from the ideal, S_{i*}, and negative-ideal, S_{i-}, alternatives. The distance from the ideal alternative is calculated as:

$$S_{i^*} = \sqrt{\sum_{j=1}^{n} (v_{ij} - v_{j^*})^2} \quad \text{for i} = 1, 2, 3, ..., \text{m}$$

The distance from the negative-ideal alternative is calculated as:

$$S_{i-} = \sqrt{\sum_{j=1}^{n} (v_{ij} - v_{j-})^2} \quad \text{for i} = 1, 2, 3, ..., \text{m}$$

6. Calculate the relative closeness to the ideal solution. The relative closeness of an alternative A_i with respect to the ideal solution A^* is calculated as:

$$MCRRM_{i*} = \frac{S_{i-}}{S_{i*} + S_{i-}}$$

Where, $1 \geq MCRRM_{i*} \geq 0$ and i = 1, 2, 3, …, m. $MCRRM_{i*}$ values close to 1 indicate better alternative performance.

The mission critical release readiness methodology's use of TOPSIS will use three alternatives (m = 3): the ideal solution; the negative ideal solution, and the current release. The criterion are the inputs to the analyze software release decision sub-function and there will be a minimum of seven criterion ($n \geq 7$). The criterion number is a minimum due to the number of resources and the software release process inputs may vary from program to program. The actual number of inputs will depend upon the program's desires. Allowing customization of the mission critical release readiness methodology broadens the application of the methodology and assists in lowering the software methodology and tools risk item. The set of weights, W, are still required to complete the TOPSIS calculations.

Weighting can be accomplished via an equal weighting system, where in the case of seven inputs, each input is weighted the same; meaning the weighting would be 0.14285 for each input and the seven weightings total to approximately one. Weighting can also be accomplished via a disciplined trade study process called Analytic Hierarchy Process (AHP) [73], [78]. Analytic Hierarchy Process was developed by T.L. Saaty and is a systematic procedure that allows pairwise comparison while prioritizing items [77],

[80]. Due to its analytical basis and widespread use, Analytic Hierarchy Process will be used to calculate the weighting of the criterion.

The Analytic Hierarchy Process steps are described below [79]:

1. Define the problem and determine the kind of knowledge sought.
2. Structure the decision hierarchy from the top with the goal of the decision, then the objectives from a broad perspective, through the intermediate levels (criteria on which subsequent elements depend) to the lowest level (which usually is a set of the alternatives).
3. Construct a set of pairwise comparison matrices. Each element in an upper level is used to compare the elements in the level immediately below with respect to it.
4. Use the priorities obtained from the comparisons to weigh the priorities in the level immediately below. Do this for every element. Then for each element in the level below add its weighted values and obtain its overall or global priority. Continue this process of weighting and adding until the final priorities of the alternatives in the bottom most level are obtained.

Comments on the Analytic Hierarchy Process steps, as applied to the mission critical release readiness methodology are as follows:

1. The big-problem definition is assisting the software release manager in deciding when to release software. The AHP problem definition is

developing a weighting methodology to weight the mission critical release readiness methodology's inputs.

2. The decision hierarchy for the mission critical release readiness methodology is one level, with the methodology inputs all at the same level.
3. There is only one level and thusly, one pairwise comparison matrix for the mission critical release readiness methodology.
4. For the mission critical release readiness methodology, there is only one level and AHP is being used to develop the weighting of the inputs against one another, not select an input (considered an alternative by AHP) over another.

Step 3 in Analytic Hierarchy Process uses a scale and their reciprocals to pairwise compare the variables under analysis. Saaty's rating scale is shown in Table 10 [79]. The scale allows the user to indicate one variable's importance over another variable by inputting the whole number in the lower left half of the pairwise matrix and the reciprocal in the upper right half of the matrix.

Table 10 – Saaty's Analytic Hierarchy Process Rating Scale

Intensity of Importance	*Definition*	*Explanation*
1	Equal Importance	Two activities contribute equally to the objective
2	Weak or slight	
3	Moderate importance	Experience and judgement slightly favour one activity over another
4	Moderate plus	
5	Strong importance	Experience and judgement strongly favour one activity over another
6	Strong plus	
7	Very strong or demonstrated importance	An activity is favoured very strongly over another; its dominance demonstrated in practice
8	Very, very strong	
9	Extreme importance	The evidence favouring one activity over another is of the highest possible order of affirmation
Reciprocals of above	If activity *i* has one of the above non-zero numbers assigned to it when compared with activity *j*, then *j* has the reciprocal value when compared with *i*	A reasonable assumption
1.1–1.9	If the activities are very close	May be difficult to assign the best value but when compared with other contrasting activities the size of the small numbers would not be too noticeable, yet they can still indicate the relative importance of the activities.

The analytical process behind the Analytic Hierarchy Process is described below [79], [81]:

Given an AHP pairwise comparison matrix, A:

$$\begin{array}{c} \\ A_1 \\ \vdots \\ A_n \end{array} \begin{array}{c} \begin{array}{ccc} A_1 & \cdots & A_n \end{array} \\ \begin{bmatrix} \frac{w_1}{w_1} & \cdots & \frac{w_1}{w_n} \\ \vdots & \ddots & \vdots \\ \frac{w_n}{w_1} & \cdots & \frac{w_n}{w_n} \end{bmatrix} \end{array}$$

The variables under analysis are $A_1, \ldots, A_n$ and the matrix entries are the absolute importance of one variable to that of another variable. There exists a scale w that can be found using the following:

$$\begin{bmatrix} \frac{w_1}{w_1} & \cdots & \frac{w_1}{w_n} \\ \vdots & \ddots & \vdots \\ \frac{w_n}{w_1} & \cdots & \frac{w_n}{w_n} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = n \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

or,

$$Aw = nw$$

Given, I the identity matrix:

$$(A - nI)w = 0$$

If n is an eigenvalue of A, then $A - nI$ vanishes and w is the eigenvector of the matrix A and the absolute importance of the variables from matrix A.

Using Analytic Hierarchy Process the weighting of the criterion can be calculated and using TOPSIS, a software release readiness metric can be calculated based on the criterion, their weightings, and the current release, non-ideal, and ideal alternatives, as shown in Figure 30. The complete mission critical release readiness methodology functional flow block diagram is shown in Figure 31.

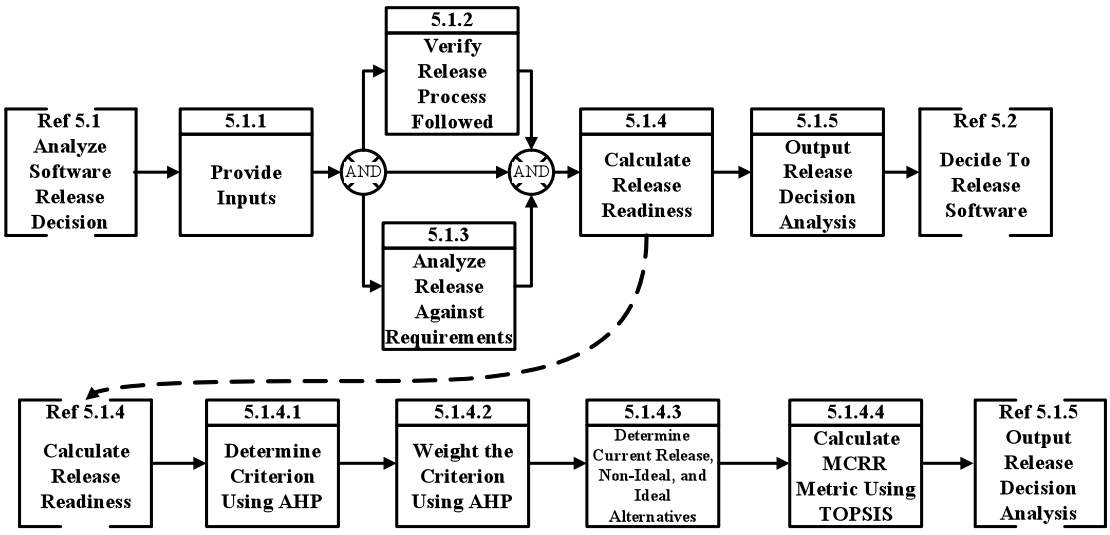

Figure 30 – Calculate Release Readiness

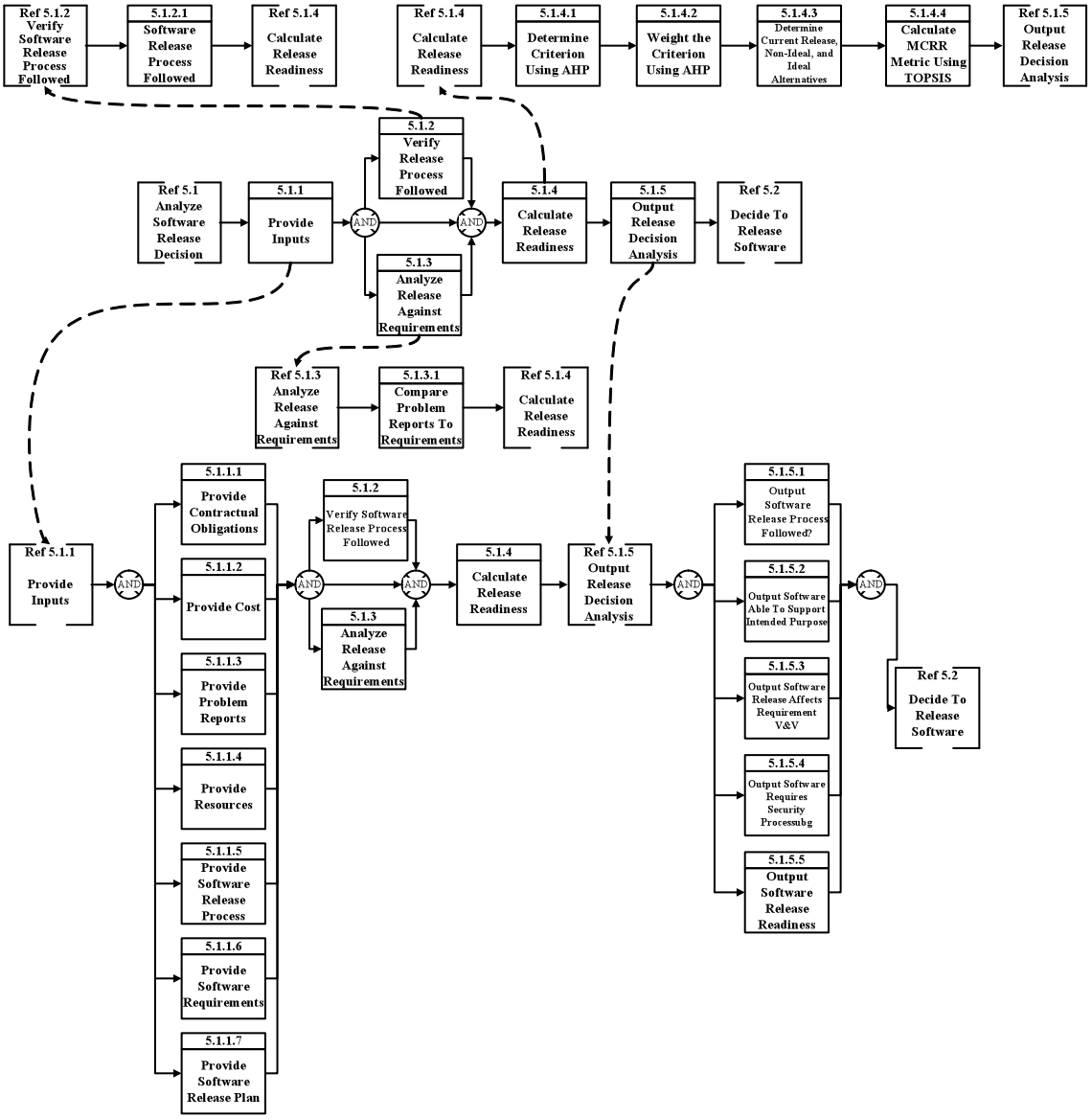

Figure 31 – Complete Mission Critical Release Readiness Methodology

The next step in the mission critical release readiness methodology development is to develop a prototype for the user interface. The user interface will not only display the software release readiness metric to the software release manager, but also allow the methodology to use the required inputs and present the required outputs of the methodology.

6.6 User Interface

The output of the mission critical release readiness methodology is critical to the successful application of the methodology. In order to assist the software release manager in the software release decision, the output of the methodology must be easily accessible via ubiquitous methods. Due to the accessibility, types, and uses of the methodology outputs, the decision was made to use Excel® to proof the methodology and its user interface. Using Excel® also provides a platform for performing the required analysis and computations during the development of the mission critical release readiness methodology. The user interface for the mission critical release readiness methodology consists of three distinct parts: 1) Analytic Hierarchy Process analysis on inputs; 2) TOPSIS analysis to calculate the mission critical release readiness metric; and 3) Output. The following sections discuss the mission critical release readiness user interfaces in detail.

6.6.1 Analytic Hierarchy Process User Interface

The mission critical release readiness methodology Analytic Hierarchy Process user interface is used to collect the project ranking on the methodology inputs and to weight the inputs for use in the TOPSIS analysis. Excel® was used to develop the Analytic Hierarchy Process user interface and the methodology was developed using Visual Basic®. The Analytic Hierarchy Process user interface allows the user to enter the pairwise rankings in the interface and then run a macro to calculate the matrix normalized eigenvectors which are the criterion weightings. The interface is shown in Figure 32. The user inputs the pairwise comparisons in the lower, left hand corner of the

matrix shaded in yellow. The blue shaded areas are the criterion names on the left hand and top side and the inverse to the user input from the yellow shaded area. The black shaded area is the diagonal and is set to 1 for the user. The tan shaded area is where the calculated normalized weighting is written.

Version: 0.0.0 4/18/2010

Mission Critical Release Readiness Methodology (MCRRM) Analytic Hierarchy Process (AHP) Worksheet	Contractual Obligations	Cost	Problem Reports	Resources: Hardware	Resources: Laboratories	Resources: Personnel	Software Release Process	Software Requirements	Software Release Plan	Normalized Weighting
Contractual Obligations	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.1111
Cost	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.1111
Problem Reports	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.1111
Resources: Hardware	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.1111
Resources: Laboratories	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.1111
Resources: Personnel	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.1111
Software Release Process	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.1111
Software Requirements	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.1111
Software Release Plan	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.1111

Figure 32 – Analytic Hierarchy Process User Interface

The Analytic Hierarchy Process calculations involve matrix multiplication, totaling rows, and normalizing with the row totals. The eigenvector of the matrix was found using the power method [82]. The power method involves sequentially squaring the matrix and calculating a normalized eigenvector until the eigenvector matches, to some degree of accuracy, the previous eigenvector. The matrix multiplication formula is shown below:

$$B_{(i,j)} = \sum_{k=1}^{n} A_{ik} A_{kj}$$

Where, i = 1, 2, 3, …, n and indicates the row; j = 1, 2, 3, …, n and indicates the column.

The source code for the matrix multiplication is shown below:

```
'Loop through, multiplying the A matrix times itself
  For i = 1 To n
    For j = 1 To n
      For k = 1 To n
        'Multiply A times A and store in B
        B(i, j) = B(i, j) + A(i, k) * A(k, j)
      Next k
    Next j
  Next i
```

The source code for the Analytic Hierarchy Process calculation is presented in Appendix A.

6.6.2 TOPSIS User Interface

The mission critical release readiness methodology TOPSIS user interface is shown in Figure 33 – MCCRM TOPSIS Worksheet UI. Similar to the Analytic Hierarchy Process user interface, the yellow shaded area is where the user inputs the criterion objective (maximize or minimize) and the data regarding the current, non-ideal,

and ideal alternatives and the blue shaded area includes the criterion and now the alternatives. The weightings, shaded in tan, are taken from the AHP worksheet. The mission critical release readiness metric is calculated using the TOPSIS methodology developed in Visual Basic® and is shown as shaded in green. Metrics are similarly provided for the negative-ideal and ideal alternatives shaded in tan.

Version: 0.0.0 4/18/2010

Mission Critical Release Readiness Methodology (MCRRM) Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) Worksheet		Criterion									MCRRM Metric
		Contractual Obligations	Cost	Problem Reports	Resources: Hardware	Resources: Laboratories	Resources: Personnel	Software Release Process	Software Requirements	Software Release Plan	
Weighting		0.1111	0.1111	0.1111	0.1111	0.1111	0.1111	0.1111	0.1111	0.1111	
Criterion Objective		Maximize	Minimize	Minimize	Maximize	Maximize	Maximize	Maximize	Maximize	Minimize	
Alternatives	Current	1	0	0	1	1	1	1	1	0	1
	Negative-Ideal	0	1	1	0	0	0	0	0	1	0
	Ideal	1	0	0	1	1	1	1	1	0	1

Figure 33 – MCCRM TOPSIS Worksheet UI

Again, the TOPSIS calculations were developed in Visual Basic® and once the data is input to the user interface, a macro is run to perform the calculations and fill in the metric data. The TOPSIS source code is presented in Appendix A.

A user interface was developed for TOPSIS criterion data entry. The software release plan user interface includes data regarding the software to be released, the pre-release testing required, the build time required to release the software, the resources required and effort to release the software, the planned and calculated software release

days, and the planned and calculated software release cost. The software release plan user interface is shown in Figure 34.

4/19/2010

Software Release Plan			
Software Release Version			
System			
Subsystem			
Function			
SLOCs			
Personnel			
SW Development Effort (Days)			
Pre-Release Test Effort (Days)			
Software Build Effort (Days)			
Resources: Laboratories			
Laborties Required			
Laborties Available			
Lab - TOPSIS Worksheet Entry			
Resources: Hardware			
Hardware Required			
Number Required			
Hardware Available			
HW - TOPSIS Worksheet Entry			
Resources: Personnel			
Number Personnel Required			
Number Personnel Available			
Release Testing			
Test Effort (Days)			
Release Schedule			
Project Start Date			
Expected Release			
Calculated Release Date			
Release Cost			
Personnel Cost Per Hour			
Expected Release Cost			
Calculated Release Cost			

Figure 34 – Software Release Plan User Interface

The calculated release date is a computational entry derived using: the project starting date; adding the maximum software development time added with the appropriate pre-release test time and software build time; and adding the software release test effort. The calculated release date formula is given below:

$$D = \max \sum_{i=1}^{n} (t_d + t_{bi} + t_{ti})$$

Where:

i = 1, …, n is the subsystem software being released;

t_d is the time to develop the subsystem software;

t_{bi} is the time to build the subsystem software;

t_{ti} is the time to pre-release test the subsystem software;

The calculated release cost is a computational entry that multiplies the number of pre-release testing times the subsystem's personnel, adds the software build time – assumes one personnel required to build the software, and adds the release testing effort times the required release personnel all times the personnel cost per hour. The formula is given below:

$$C = [t_r \times p_r + \sum_{i=1}^{n} (t_{bi} + (t_{ti} \times p_i))] * c_{ph}$$

Where:

i = 1, …, n is the subsystem software being released;

t_{bi} is the time to build the subsystem software;

t_{ti} is the time to pre-release test the subsystem software;

t_r is the time to release test the software;

p_i is the subsystem personnel required for pre-release testing;

p_r is the personnel required for release testing;

c_{ph} is personnel cost per hour.

The software release plan user interface is the source of several inputs on the TOPSIS worksheet. The resources: laboratories and resources: hardware are input onto the TOPSIS worksheet as binary results: 1 – indicates there are resources available for the software release; 0 – indicates there are not enough required resources available for the software release. The resources: personnel – number of personnel required is a direct input to the ideal alternative entry on the TOPSIS worksheet and the resources: personnel – number of personnel available is directly input to the current alternative entry. Release schedule – expected release date is a direct entry of 0 into the ideal alternative entry on the TOPSIS worksheet and the release schedule – calculated release date, is the integer days difference between expected release date and calculated release date and is directly input into the current alternative entry. The ideal alternative entry on the TOPSIS worksheet for cost is a direct entry from the release cost – expected release cost and the current entry for cost is directly from the release cost – calculated release cost.

6.6.3 Output User Interface

The mission critical release readiness methodology output user interface – mission critical release readiness dashboard – is shown in Figure 35. Although the TOPSIS worksheet and the dashboard share the same input data, the dashboard is used to present the mission critical release readiness methodology results. The dashboard presents the expected and current/projected data, the mission critical release readiness metric, and answers the four questions required of the methodology, with color enhancement, without adding the TOPSIS required data of weighting, objectives, negative ideal and ideal alternatives.

4/18/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations		
Cost		
Problem Reports		
Resources: Hardware		
Resources: Laboratories		
Resources: Personnel		
Software Release Process		
Software Requirements		
Software Release Plan		

MCRRM

Software Release
Software Release Affects V&V?
Software Release Process Followed?
Software Requires Security Processing?
Software Able To Perform Intended Purpose?

Figure 35 – Dashboard User Interface

The dashboard user interface also allows the project to program the interface to indicate the goodness of the expected data against current data and to program a color indicator for the mission critical release readiness metric. An example of cost data and the mission critical release readiness metric exceeding project selected thresholds and indicated in red is shown in Figure 36.

4/18/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$28,140.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.8107

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 36 – Dashboard Color Coding Example

The color coding gives an additional indication of areas of concern for the software release under consideration. A dashboard item with its Current/Projected Value background color: Green indicates no concerns; Yellow indicates slight concern; Red indicates a major concern. The color-coding is project controlled and in the example shown in Figure 37, the project chose to use a ratio of expected to current/projected values as the key indicator. The ratio values shown with a gray background indicate maximize values, the light blue background indicate the minimize values, and the light green background indicate discrete values. The conditional formatting option in Excel®

was used to color codes the dashboard Current/Projected Values with the values shown in the Green, Yellow, and Red columns.

9/16/2010

MCRRM Dashboard	0.94 Green	0.86 Yellow	0.94 Yellow	0.86 Red	Current Release Expected Value	Current/ Projected Value
Contractual Obligations	1	0.95	1	0.95	1	1
Cost	1	0.92	1	0.92	$22,740.00	$22,740.00
Problem Reports	0.85	0.5	0.85	0.5	8	8
Resources: Hardware	1	0.9	1	0.9	1	1
Resources: Laboratories	1	0.9	1	0.9	1	1
Resources: Personnel	8	6	8	6	8	8
Software Release Process	1	0.95	1	0.95	1	1
Software Requirements	0.95	0.86	0.95	0.86	12	12
Software Release Plan	2/25/2010	3/3/2010	2/25/2010	3/3/2010	2/25/2010	2/25/2010
					MCRRM	1.0000

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Probable

Figure 37 – MCRRM Conditional Formatting Data

Figure 38 is an example of the conditional formatting for the mission critical release readiness metric value. The metric uses the same values as those for the Current Release headers, which are indicated as cells D3, E3, F3, and G3 in the example and the metric value is in cell J15.

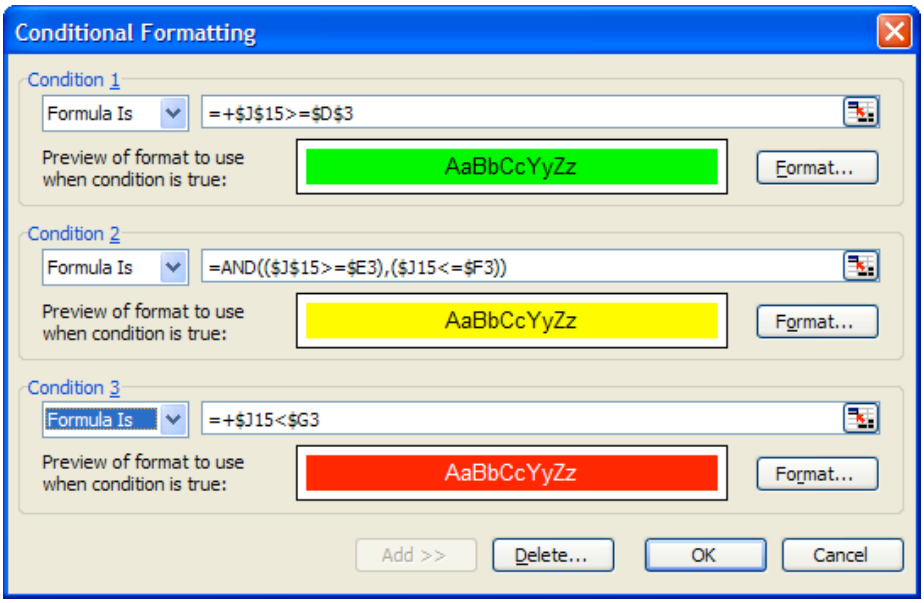

Figure 38 – Conditional Formatting Equations

In addition to providing the user with the expected value, the current/projected value, and the metric value, the mission critical release readiness methodology output user interface includes the answers to the four questions:

- Software Release Affects V&V?
- Software Release Process Followed?
- Software Requires Security Processing?
- Software Able To Perform Intended Purpose?

The first question (Software Release Affects V&V?) is an output of the analyze release against requirements sub-function. The analyze release against requirements sub-function's methodology compares the problem reports and the functions they affect

against the requirements and the functions they implement. The methodology will factor the problem report's severity and the level of verification and validation of the requirement in determining whether the software release affects verification and validation activities. For the proof of the mission critical release readiness methodology, the analysis of problem reports and requirements is accomplished and any requirements with severity 1 or 2 problem reports written against the requirement's implemented functions are highlighted. The question regarding the software release affecting verification and validation is then answered based on the results of the analysis.

The next questions are regarding the software release process and whether the release requires security processing. Both of these questions will be answered via yes/no responses. Modeling the project specific software release process and automating the answering of whether the software release process is followed is left for future work of the mission critical release readiness methodology.

The last question is regarding the ability of the software to support its intended purpose. This question will be answered by one of three responses: Highly Probable; Probable; Improbable. The answer will be derived through analysis of the mission critical release readiness metric, analyzing the software affects verification and validation response, analyzing whether the software release process was followed, and analyzing the stability of the software.

Software stability analysis is accomplished by comparing the previous 6 weeks moving average of problem reports per week versus the current 6 weeks moving average of problem reports per week. The three responses and their determinations are shown below:

- Highly Probable – Software release process followed, mission critical release readiness metric is above 0.94, no warnings or cautions indicated in the metric calculation, and the current 6 weeks moving average of problem reports per week is less than or equal to 110% of the previous 6 weeks moving average of problem reports per week.
- Probable – Software release process followed, mission critical release readiness metric is between 0.86 and 0.94 with no warnings indicated in the metric calculation, and the current 6 weeks moving average of problem reports per week is less than 125% of the previous 6 weeks moving average of problem reports per week.
- Improbable – Software release process not followed, mission critical release readiness metric below 0.86, software release affects V&V, and/or the current 6 weeks moving average of problem reports per week is greater than 125% of the previous 6 weeks moving average of problem reports per week.

The data for the software release will be analyzed and using the guidelines above, the software's ability to support its intended purpose question will be answered.

6.7 Developing The Methodology Summary

The mission critical release readiness methodology's analytical methodologies and user interfaces are developed after a review of the methodology's inputs and

producing the functional flow block diagram for the methodology. The methodology inputs are weighted using Analytic Hierarchy Process and the weightings are then used as part of the TOPSIS methodology for developing the mission critical release readiness metric. Excel® is used to develop the Analytic Hierarchy Process and TOPSIS worksheets and analyses with the source code provided in Appendix A.

The user interface for the mission critical release readiness methodology is developed in Excel® allowing for development of both a user interface and analytic computational procedures. Three user interfaces are developed: 1) Analytic Hierarchy Process analysis on inputs; 2) TOPSIS analysis to calculate the mission critical release readiness metric; and 3) Output. The user interfaces are developed and the interfaces and specific features are provided.

The analytical process behind the mission critical release readiness methodology and the calculation of the mission critical release readiness metric is provided in detail, as is the calculation and presentation of the methodology's required outputs. With the development of the methodology complete, verification of the methodology's analytical processes and verification and validation that the methodology meets the software release manager's needs and requirements is required. Verification and validation of the mission critical release readiness methodology is covered in the next chapter.

Chapter 7

VERIFICATION AND VALIDATION OF METHODOLOGY

7.1 Verification

The *Verification* sub-process is where formal testing and evaluation are performed on the developed product [17]. The test results are used to evaluate the capability of the system under development, to adequately meet the requirements of the system, verifying the system works as intended. The test results are also used to evaluate the system's ability to meet the needs of the user, validating the system.

7.2 Verify And Validate Solution

In the need identification and problem resolution process' *Verify and Validate Solution* sub-process, the system under development is tested, the test results are used to verify the system requirements, and finally, the product is validated using the test results to insure the system satisfies the customer's needs.

7.3 Analytic Hierarchy Process Verification

The development of the Analytic Hierarchy Process user interface for the mission critical release readiness methodology requires verification of the developed analytical

methodology. The Analytic Hierarchy Process analytical methodology was verified in two steps.

First, the matrix multiplication code was verified against the MMult function in Excel® for 2 by 2, 3 by 3, 5 by 5, and 9 by 9 matrixes. The matrix multiplication developed and coded for the Analytic Hierarchy Process analytical methodology, exactly matched the MMult function.

Second, the eigenvector calculations were verified using MATLAB®, Mathematica, and an online matrix calculator [83]. The eigenvector calculated using the power method for the mission critical release readiness methodology matched the MATLAB®, Mathematica, and online matrix calculator results, upon normalization, out to the fifth decimal place.

With the matrix multiplication and eigenvector calculations verified, the mission critical release readiness methodology developed Analytic Hierarchy Process analytical methodology passes verification.

7.4 TOPSIS Methodology Verification

The development of the TOPSIS user interface for the mission critical release readiness methodology requires verification of the developed TOPSIS analytical methodology. The TOPSIS analytical methodology was verified dynamically by inputting data and insuring the analytical methodology produced logical results and by comparison with a commercial TOPSIS calculator. For dynamic testing, the developed TOPSIS analytical methodology's outputs were verified against known inputs to insure

the developed methodology was creating logical results. The dynamic testing indicated the TOPSIS analytical methodology produced expected results for known inputs.

The Statistical Design Institute (SDI) produces a suite of software that includes a TOPSIS calculator [84]. The mission critical release readiness developed TOPSIS analytical methodology was compared against the SDI TOPSIS calculator and the developed analytical methodology's output matched the output of the SDI toolset.

With the TOPSIS analytical methodology passing dynamic and comparison testing, the mission critical release readiness methodology developed TOPSIS analytical methodology passes verification.

7.5 Methodology Verification

Verification of the mission critical release readiness methodology is accomplished via test and inspection, per the methodology requirements verification methods detailed in Table 3 and Table 4. The mission critical release readiness requirements verified by test are discussed first. The inspection method of verification is documented in a Verification Cross Reference Matrix (VCRM). The verification cross reference matrix documents the requirements, the requirement verification methods, and documents the verification of the requirement. Two verification cross reference matrixes are provided, one for the mission critical release readiness requirements and one for the mission critical release readiness methodology derived requirements.

7.5.1 Methodology Testing

Mission critical release readiness methodology testing is accomplished via dynamic testing – performing case studies using the methodology and insuring the methodology produces logical results. A generalized testing case study is presented next.

7.5.1.1 Case Study: Release Plan User Interface

The software release plan user interface for the case study is shown in Figure 39. The mission critical release readiness methodology is applied on the 2/25/2010, the day the software is to be released, to provide the software release manager an analytical basis for assisting in deciding to release the software and its ability to perform its intended purpose. The software release plan user interface provides the data for software release version 0.9.1, which consists of a system generically noted as S1 and three subsystems generically noted as SS1, SS2, and SS3, implementing function A1. The software release consists of 10,000 source lines of code distributed over the three subsystems.

Subsystem SS1 is developing 3000 source lines of code, estimating to take 100 days with 3 people, pre-release testing will take 3 days at 8 hours a day, and 0.5 day is required to compile and build the software.

Subsystem SS2 develops 6300 source lines of code, estimating to take 180 days with 4 people, pre-release testing will be 5 days, and 1 day is required to compile and build the software.

Subsystem SS3 develops 700 lines of code, estimating to take 30 days with 1 person, pre-release testing will be 1 day, and 0.1 day is required to compile and build the software.

2/25/2010

Software Release Plan			
Software Release Version	0.9.1		
System	S1		
Subsystem	SS1	SS2	SS3
Function	A1	A1	A1
SLOCs	3000	6300	700
Personnel	3	4	1
SW Development Effort (Days)	100	180	30
Pre-Release Test Effort (Days)	3	5	1
Software Build Effort (Days)	0.5	1	0.1
Resources: Laboratories			
Laboratories Required	L1	L2	L2
Laboratories Available	Yes	Yes	Yes
Lab - TOPSIS Worksheet Entry	1		
Resources: Hardware			
Hardware Required	SS1	SS2	SS3
Number Required	2	2	1
Hardware Available	Yes	Yes	Yes
HW - TOPSIS Worksheet Entry	1		
Resources: Personnel			
Number Personnel Required	8		
Number Personnel Available	8		
Release Testing			
Test Effort (Days)	15		
Release Schedule			
Project Start Date	8/8/2009		
Expected Release	2/25/2010		
Calculated Release Date	2/25/2010		
Release Cost			
Personnel Cost Per Hour	$150.00		
Expected Release Cost	$22,740.00		
Calculated Release Cost	$22,740.00		

Figure 39 – Case Study SW Release Plan

The resources required to release software version 0.9.1 are as follows: two laboratories; five subsystems (2 SS1, 2 SS2, and 1 SS3); and eight personnel. All required resources are available to support software version 0.9.1 release.

The software version 0.9.1 project started on 8/8/2009. Development of the software was expected to take 180 days. Pre-release testing and compile and build activities were expected to take 6 days and release testing of the total system was expected to take 15 days. The expected release date is 2/25/2010, with a calculated release date of 2/25/2010. The calculated release date is calculated as follows:

$$D = \max \sum_{i=1}^{n} (t_d + t_{bi} + t_{ti})$$

Where:

i = 1, …, n is the subsystem software being released;

t_d is the time to develop the subsystem software;

t_{bi} is the time to build the subsystem software;

t_{ti} is the time to pre-release test the subsystem software;

The calculated release cost of $22,740 equals the expected release cost. The calculated release cost formula is as follows:

$$C = [t_r \times p_r + \sum_{i=1}^{n} (t_{bi} + (t_{ti} \times p_i))] \times c_{ph}$$

Where:

i = 1, …, n is the subsystem software being released;

t_{bi} is the time to build the subsystem software;

t_{ti} is the time to pre-release test the subsystem software;

t_r is the time to release test the software;

p_i is the subsystem personnel required for pre-release testing;

p_r is the personnel required for release testing;

c_{ph} is personnel cost per hour.

The case study calculated release cost is calculated as below:

$$\$22{,}740 = [15 \times 8 + [0.5 + (3 \times 3)] + [1 + (5 \times 4)] + [0.1 + (1 \times 1)]] \times 150$$

The software release plan user interface provides a overview of the software under consideration for release, next the Analytic Hierarchy Process worksheet is presented for the case study.

7.5.1.2 Case Study: Analytic Hierarchy Process Worksheet

The Analytic Hierarchy Process worksheet for the case study is shown in Figure 40. Inputs from the case study were input in the lower left section of the worksheet highlighted in yellow. The inverse number was automatically filled in by the worksheet in the corresponding upper right. Upon entering all the data in the yellow section, the developed Analytic Hierarchy Process code was ran and produced the calculated normalized weightings shown with the tan highlights. The normalized weightings

indicate contractual obligations and cost are the two highest weighted criterions followed by software release plan and resources: personnel.

Version: 0.9.1 2/25/2010

Mission Critical Release Readiness Methodology (MCRRM) Analytic Hierarchy Process (AHP) Worksheet	Contractual Obligations	Cost	Problem Reports	Resources: Hardware	Resources: Laboratories	Resources: Personnel	Software Release Process	Software Requirements	Software Release Plan	Normalized Weighting
Contractual Obligations	1.000	1.000	9.000	7.000	7.000	3.000	9.000	3.000	2.000	0.2667
Cost	1.000	1.000	9.000	7.000	7.000	3.000	9.000	3.000	2.000	0.2667
Problem Reports	0.111	0.111	1.000	0.143	0.143	0.333	0.333	0.333	0.200	0.0198
Resources: Hardware	0.143	0.143	7.000	1.000	1.000	0.200	0.333	0.333	0.333	0.0407
Resources: Laboratories	0.143	0.143	7.000	1.000	1.000	0.200	0.333	0.333	0.333	0.0407
Resources: Personnel	0.333	0.333	3.000	5.000	5.000	1.000	1.000	1.000	1.000	0.1016
Software Release Process	0.111	0.111	3.000	3.000	3.000	1.000	1.000	1.000	0.333	0.0653
Software Requirements	0.333	0.333	3.000	3.000	3.000	1.000	1.000	1.000	0.500	0.0792
Software Release Plan	0.500	0.500	5.000	3.000	3.000	1.000	3.000	2.000	1.000	0.1193

Figure 40 – Case Study AHP Worksheet

The normalized weightings from the Analytic Hierarchy Process worksheet are used by the TOPSIS worksheet and using the magic of Excel® are automatically copied into the TOPSIS worksheet in the weighting row. The case study TOPSIS worksheet is shown in the next section.

7.5.1.3 Case Study: TOPSIS User Interface

The inputs for the TOPSIS worksheet come from several different sources. The weightings are calculated in the Analytic Hierarchy Process worksheet and copied directly from the Analytic Hierarchy Process worksheet to the weightings inputs on the TOPSIS worksheet. The criterion objectives are input by the project and the case study

objectives are to maximize all criterion except cost, problem reports, and software release plan which are minimized.

The contractual obligations are input by the project with a 1 indicating the contractual obligations will be met and a 0 indicating they will not be met – the case study's contractual obligations are being met, therefore a 1 is input in the current and ideal alternative entries and a 0 in the negative ideal alternative. The cost inputs for the current and ideal alternatives are input from the case study's software release plan user interface and the negative ideal cost input for the case study is about 10% greater than the ideal cost. The problem reports inputs are input by the project using the project's problem report data – Table 11 shows the relevant case study problem reports. The case study's problem reports ideal alternative input 8 and the current alternative of 10 is the total number of open problem reports against the function A1. The case study chose 25 for the negative ideal problem report input.

Table 11 – Case Study Relevant Problem Reports

2/25/2010

Problem Reports										
PRs Current Week		8	Previous 6 Wks Avg PR/Wk			11	6 Wks Avg PR/Wk With Current			10.2
Ident	Problem	Severity	Status	SW Version Affected	System	Sub-System Affected	Function Affected	Date Created	Date Closed	SW Version Implemented In
PR101	...	3	Open	0.9.0	S1	SS1	A1	1/22/2010		
PR102	...	2	Closed	0.9.0	S3	SS3	A3	1/25/2010	2/4/2010	0.9.1
PR103	...	4	Closed	0.9.0	S1	SS3	A1	1/25/2010	2/5/2010	0.9.1
PR104	...	4	Open	0.9.0	S3	SS3	A4	1/27/2010		
PR105	...	5	Open	0.9.0	S1	SS1	A1	1/27/2010		
PR106	...	3	Open	0.9.1	S1	SS3	A1	2/2/2010		
PR107	...	4	Open	0.9.1	S1	SS2	A1	2/2/2010		
PR108	...	4	Open	0.9.1	S1	SS2	A1	2/2/2010		
PR109	...	5	Open	0.9.0	S3	SS3	A1	2/3/2010		
PR110	...	3	Open	0.9.1	S1	SS1	A1	2/4/2010		
PR111	...	3	Open	0.9.1	S1	SS2	A1	2/4/2010		
PR112	...	4	Open	0.9.1	S1	SS2	A1	2/4/2010		
PR113	...	3	Open	0.9.1	S1	SS2	A1	2/5/2010		

The current and ideal alternative resources inputs are all input directly from the software release plan user interface. The negative alternative inputs for hardware, laboratories, and personnel a well as software release process are all 0.9. The current and ideal alternative inputs for software release process are both 1 indicating the process is being followed for the case study. There are 12 software requirements for the case study's 0.9.1 version release and all 12 are able to be verified. The current and ideal alternatives inputs for software requirements are both 12 with the negative ideal input being 5. The software release plan inputs for current and ideal alternative are input directly from the software release plan user interface and both are 0 representing 0 days away from the planned release date of 2/25/2010 for the case study. The negative ideal input is 6 days away from the planned release date of 2/25/2010 or 3/3/2010 for the case study. TOPSIS user interface for the case study is shown in Figure 41.

Version: 0.9.1 2/25/2010

Mission Critical Release Readiness Methodology (MCRRM) Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) Worksheet		Criterion									MCRRM Metric
		Contractual Obligations	Cost	Problem Reports	Resources: Hardware	Resources: Laboratories	Resources: Personnel	Software Release Process	Software Requirements	Software Release Plan	
Weighting		0.2667	0.2667	0.0198	0.0407	0.0407	0.1016	0.0653	0.0792	0.1193	
Criterion Objective		Maximize	Minimize	Minimize	Maximize	Maximize	Maximize	Maximize	Maximize	Minimize	
Alternatives	Current	1.0	$22,740	10.0	1.0	1.0	8.0	1.0	12.0	0.00	0.9901
	Negative-Ideal	0.9	$25,000	25.0	0.9	0.9	0.9	0.9	5.0	6.00	0
	Ideal	1.0	$22,740	8.0	1.0	1.0	8.0	1.0	12.0	0.00	1

Figure 41 – Case Study TOPSIS Worksheet

The TOPSIS worksheet inputs are used to generate the mission critical release readiness methodology dashboard for the software under release consideration. The dashboard for the case study is shown in the next section.

7.5.1.4 Case Study: Dashboard

The data from the case study's TOPSIS worksheet is copied directly to the mission critical release readiness methodology dashboard. The dashboard for the case study is shown in Figure 42. The dashboard indicates that although the mission critical release readiness metric is 0.9901 and not indicating a caution or warning for the overall metric, the current/projected value of the problem reports has exceeded the case study's selected value for caution and is consequently highlighted in yellow.

2/25/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	10
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.9901

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Probable

Figure 42 – Case Study MCRRM Dashboard

The case study dashboard indicates all the questions required of the mission critical release methodology have been answered. The software release does not affect verification and validation and security processing is not required. The software release process was followed to release version 0.9.1 and due to a cautionary indication with the problem reports, the question about the software able to perform its intended duty is answered with a probable.

The case study dashboard provides an analytical metric of the release readiness of the software, a qualitative metric on the software's ability to perform its intended

purpose, and identifies areas of concern with respect to the software release – problem reports. Based on the case study's mission critical release readiness dashboard it would be recommended to release the software, with a cautionary note that the current problem reports are slightly more than the expected value.

7.5.2 Verification Cross Reference Matrixes

A verification cross reference matrix maps requirements to the verification activities that verify the requirement. By mapping the requirement to the verification activity, the matrix allows a program to status the verification activity and insures all requirements have verification activities assigned. Verification cross reference matrixes were developed for the methodology requirements and the derived implementation requirements.

The methodology requirement verification cross reference matrix is shown in Table 12. The table maps the methodology requirements to the section in this document that verifies the requirement. All methodology requirements are linked to verification activities and are considered verified.

The mission critical release readiness methodology derived implementation requirement verification cross reference matrix is shown in Table 13. The table maps the derived implementation requirements to the section in this document that verifies the requirement. All derived implementation requirements are linked to verification activities and are considered verified.

Table 12 – Requirement Verification Cross Reference Matix

Parent Ident	Req ID	Requirement Text	Verification Method	Verifying Section
N1	R1	Once the methodology has been initialized, it shall take no more than 4 hours for the methodology to produce decision analysis support results.	Test	By using Excel® and using automation through built in Excel® functions, macros, and coding portions of the mission critical release readiness methodology, upon initialization (all inputs provided) the analysis and providing decision analysis support results occur upon entry of the last data input with no recognizable delays.
N2	R2	The methodology shall indicate whether the software release follows the software release process.	Inspection	6.6.3 – The MCRRM output user interface indicates an output of the methodology is whether the release follows the software release process.
N3	R3	The methodology shall provide the date the methodology analyzed the software release.	Inspection	6.6.1, 6.6.2, & 6.6.3 – The user interfaces for the methodology all include a date field for when the analysis was accomplished.
N4	R4	The methodology shall provide a qualitative measure on the software release's ability to perform its intended purpose.	Inspection	6.6.3 – The MCRRM output user interface indicates an output of the methodology is whether the release can perform its intended purpose and how the qualitative measure is answered.
N5	R5	Software releases containing severity 1 or 2 problem reports affecting requirement V&V shall be identified.	Inspection	6.6.3 – The MCRRM output user interface indicates an output of the methodology is whether the release affects requirement V&V.
N6	R6	The methodology shall list software releases requiring additional processing or testing as part of the release process.	Inspection	6.4.1 & 6.6.3 – The generic software release process used as an input to the MCRRM has decision points for additional processing or testing and the MCRRM output user interface indicates whether the release followed the software release process.
N7	R7	The methodology shall permit the software release manager to track the cost of the software release.	Inspection	6.6.3 – The MCRRM output user interface indicates an input of the methodology is both expected and projected cost.
N8	R8	The release options requiring security processing shall be identified	Inspection	6.6.3 – The MCRRM output user interface indicates an output of the methodology is whether the release requires security processing.
N9	R9	The released software shall be capable of supporting its intended purpose.	Inspection	6.6.3 – The MCRRM output user interface indicates an output of the methodology is whether the software release is able to support its intended purpose and the calculation of the answer includes a measure of software stability.

Table 13 – Derived Requirements Verification Cross Refernce Matrix

Parent Ident	Req ID	Requirement Text	Verification Method	Verifying Section
R3	DR1	The methodology shall use contractual obligations as an input.	Inspection	6.6.3 – The MCRRM output user interface indicates an input of contractual obligations.
R3	DR2	The methodology shall use the current software release plan as an input.	Inspection	6.6.3 – The MCRRM output user interface indicates an input of the software release process.
R3	DR3	The methodology shall use the current problem reports as an input.	Inspection	6.6.3 – The MCRRM output user interface indicates an input of problem reports.
R3	DR4	The methodology shall use the current software requirements.	Inspection	6.6.3 – The MCRRM output user interface indicates an input of current software requirements.
R3	DR5	The methodology shall use current resource availability as an input.	Inspection	6.6.3 – The MCRRM output user interface indicates inputs of resources.
R3	DR6	The methodology shall use the current software release process as an input.	Inspection	6.6.3 – The MCRRM output user interface indicates an input of the software release process.
R7	DR7	The methodology shall use the cost of releasing software as an input.	Inspection	6.6.3 – The MCRRM output user interface indicates inputs of both expected and projected cost.

7.6 Sensitivity Analysis

Sensitivity analysis is the study of a system in response to changes in data and parameters [90]. Sensitivity analysis was performed on the mission critical release readiness methodology by: 1) varying the values of the criterion and studying the response of the dashboard to the values; 2) randomly varying the pairwise comparison of the criterion and studying the response of the mission critical release readiness metric; and 3) randomly varying the values of the criterion and randomly varying the pairwise comparison of the criterion and studying the response of the methodology. Figure 43 is a plot of some of the test runs accomplished during sensitivity testing. Over 11536 runs were accomplished. The criterion test points are points were the criterion was varied or the criterion and pairwise comparison varied. The combinational plot line are test points that varied more than one criterion and/or pairwise comparison, or were test cases run against other case studies.

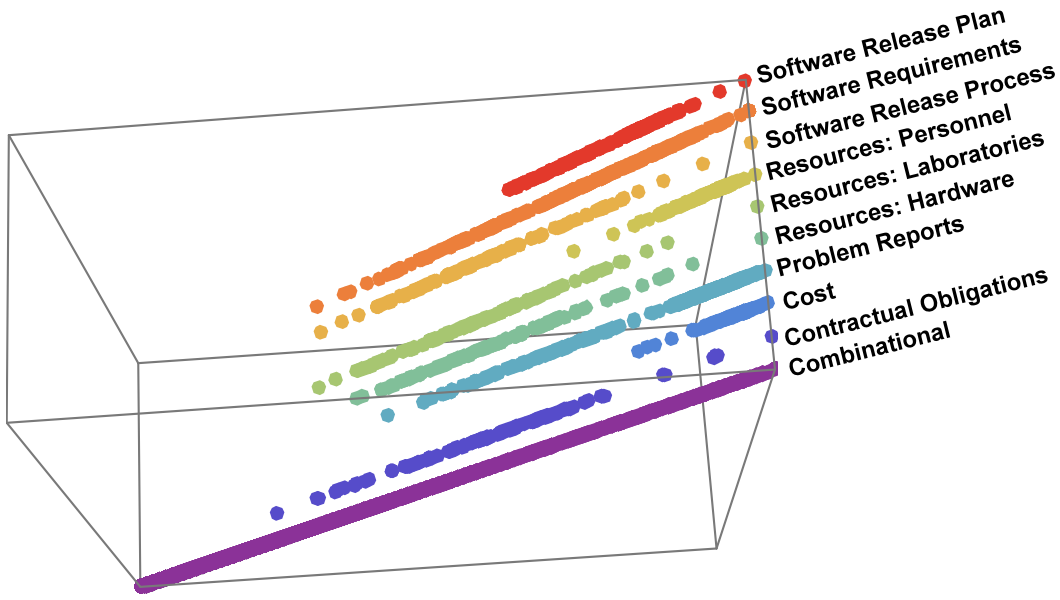

Figure 43 – Sensity Test Runs

The following sections describe the sensitivity analyses in detail.

7.6.1 Criterion Values

The criterion sensitivity analysis in this section was performed by varying the case study criterion and recording the results in the methodology's dashboard. All of the case study criterions were reset to the "Ideal" alternative values before starting a run to insure only the effect of the criterion under evaluation was analyzed. The "Resources: Personnel" criterion was chosen as an example of the analysis to include here, the remaining analysis results are shown in Appendix B. "Resources: Personnel" criterion was chosen to sweep from a higher number than the ideal alternative, down to 0 and capturing the dashboard for all changes in the criterion value. All other methodology

criterions will only capture the dashboard for the first change in the dashboard, not the entire range of criterion values.

The criterion sensitivity analysis process started with all the current criterions equal to the case study ideal alternative criterions values, as shown in Figure 44 and Figure 45. Using the ideal alternative values for the current alternative criterions meant that the current and ideal alternatives were equal to begin sensitivity analysis. Then the current criterion was decreased or increased, depending upon whether the criterion objective was to maximize or minimize, and the dashboard captured when significant levels were reached.

Version: 0.9.1 9/6/2010

Mission Critical Release Readiness Methodology (MCRRM) Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) Worksheet		Criterion									MCRRM Metric
		Contractual Obligations	Cost	Problem Reports	Resources: Hardware	Resources: Laboratories	Resources: Personnel	Software Release Process	Software Requirements	Software Release Plan	
Weighting		0.2667	0.2667	0.0198	0.0407	0.0407	0.1016	0.0653	0.0792	0.1193	
Criterion Objective		Maximize	Minimize	Minimize	Maximize	Maximize	Maximize	Maximize	Maximize	Minimize	
Alternatives	Current	1.0	$22,740	8.0	1.0	1.0	8.0	1.0	12.0	0.00	1
	Negative-Ideal	0.9	$25,000	25.0	0.9	0.9	0.9	0.9	5.0	6.00	0
	Ideal	1.0	$22,740	8.0	1.0	1.0	8.0	1.0	12.0	0.00	1

Figure 44 – Criterion Sensitivity Analysis Starting Point

2/25/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	1.0000

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Probable

Figure 45 – Dashboard Starting Point

7.6.1.1 Resources: Personnel

As a demonstration of what occurs when a maximize criterion was increased over the ideal alternative criterion value, the "Resources: Personnel" criterion was increased to 9 to start the sensitivity analysis. Figure 46 shows the effect of increasing a maximize criterion above the ideal alternative value, the current alternative becomes the ideal alternative and the methodology measures how far the ideal alternative is from the current alternative. Figure 47 is the dashboard for increasing the number of personnel to 9.

Version: 0.9.1 9/6/2010

Mission Critical Release Readiness Methodology (MCRRM) Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) Worksheet		Criterion									MCRRM Metric
		Contractual Obligations	Cost	Problem Reports	Resources: Hardware	Resources: Laboratories	Resources: Personnel	Software Release Process	Software Requirements	Software Release Plan	
Weighting		0.2667	0.2667	0.0198	0.0407	0.0407	0.1016	0.0653	0.0792	0.1193	
Criterion Objective		Maximize	Minimize	Minimize	Maximize	Maximize	Maximize	Maximize	Maximize	Minimize	
Alternatives	Current	1.0	$22,740	8.0	1.0	1.0	9.0	1.0	12.0	0.00	1
	Negative-Ideal	0.9	$25,000	25.0	0.9	0.9	0.9	0.9	5.0	6.00	0
	Ideal	1.0	$22,740	8.0	1.0	1.0	8.0	1.0	12.0	0.00	0.9431

Figure 46 – Personnel Increased to 9

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	9
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
MCRRM		1.0000

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Highly Probable

Figure 47 – Dashboard For 9 Personnel

Figure 48 is the dashboard for decreasing the "Resources: Personnel" criterion to 7. The dashboard indicates a mission critical release readiness metric value of 0.9357, indicating a cautionary value, in this case due to resources being below the level required

to support the release plan as indicated by the red highlight on the "Resource: Personnel" criterion. Although having 7 personnel to produce this release produces a cautionary metric, the metric is indicating release the software, but the actual release date may be in danger of slipping due to a shortage of the required personnel. The "Improbable" answer to the question "Software Able To Perform Intended Purpose?" is due to a warning occurring in the dashboard for too few personnel.

9/16/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	7
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.9357

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Probable

Figure 48 – Dashboard For 7 Personnel

The effect of reducing the "Resources: Personnel" to 6 is shown in Figure 49. With the personnel reduced to 6, the methodology is producing a cautionary value of 0.8705, indicating the software release may not be able to support its intended purpose, in this case the release date, at its current state of personnel.

9/16/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	6
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
MCRRM		0.8705

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Probable

Figure 49 – Dashboard For 6 Personnel

Reducing the "Resources: Personnel" criterion value to 5 produces warning metric value of 0.8057 indicating the release cannot support its intended release date, as shown in Figure 50. At values below 5, a lower mission critical release readiness metric value is derived, but will have the same consequences on the release decision, just in more severe cases. Reducing personnel values from 4 to 0 are shown in Figure 51, Figure 52, Figure 53, Figure 54, and Figure 55.

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	5
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.8057

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 50 – Dashboard For 5 Personnel

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	4
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.7430

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 51 – Dashboard For 4 Personnel

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	3
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.6846

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 52 – Dashboard For 3 Personnel

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	2
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.6328

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 53 – Dashboard For 2 Personnel

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	1
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.5895

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 54 – Dashboard For 1 Personnel

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	0
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.5550

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 55 – Dashboard For 0 Personnel

7.6.2 Pairwise Comparison

The pairwise comparison analysis was performed in multiple steps. First, the pairwise comparisons were varied randomly with fixed criterion values. Then the pairwise comparisons were varied randomly with multiple randomly varied criterion values. Finally, the pairwise comparisons were varied randomly while randomly varying all the criterion.

7.6.2.1 Random Pairwise Comparisons And Fixed Criterion

The pairwise comparison sensitivity analysis was performed by inputting random pairwise comparison values into the Analytic Hierarchy Process spreadsheet, running the AHPCalc macro to calculate the weightings, transferring the weightings to the TOPSIS spreadsheet, running the TOPSISCalc macro to calculate the mission critical release readiness metric, storing the random and metric values, and then performing statistical analysis on the resulting metric values. The pairwise comparison sensitivity analysis was repeated 100 times for every test using Excel and visual basic code. The code for the test is provided below:

```
'*********************************************
'*
'*  AHPSense randomly assigns values to the
'*  AHP spreadsheet, calculates the criterion
'*  weighting based on the random data, calculates
'*  the TOPSIS value for the new AHP weightings,
'*  and stores the random assignments and TOPSIS
'*  results in the "Sensitivity.RND" spreadsheet.
'*
'*  Author:  Tim Woods          September 2010
'*
'*********************************************
```

```
Sub AHPSense()

    'car is an array to hold the number
    'individual inputs to insure even
    'distribution
    Dim car(17) As Double
    Dim c As Double

    'Run 100 times
    For s = 1 To 100

        'Row and column starting points
        rw = 5
        cl = 3

        'Initialize counter
        cnt = 0

        'c is 1/17 for the number of AHP entries
        c = 0.05882353

        'n*n data entries for AHP
        'Upper Right and diagonal taken care of
        For i = 1 To ((n * n) - n) / 2

            'Random data
            t = Rnd()

            'Use Random data to input ranking
            Select Case t

                Case Is <= c
                va = 1 / 9
                car(1) = car(1) + 1

                Case Is <= 2 * c
                va = 1 / 8
                car(2) = car(2) + 1

                Case Is <= 3 * c
                va = 1 / 7
                car(3) = car(3) + 1

                Case Is <= 4 * c
                va = 1 / 6
```

```
car(4) = car(4) + 1

Case Is <= 5 * c
va = 1 / 5
car(5) = car(5) + 1

Case Is <= 6 * c
va = 1 / 4
car(6) = car(6) + 1

Case Is <= 7 * c
va = 1 / 3
car(7) = car(7) + 1

Case Is <= 8 * c
va = 1 / 2
car(8) = car(8) + 1

Case Is <= 9 * c
va = 1
car(9) = car(9) + 1

Case Is <= 10 * c
va = 2
car(10) = car(10) + 1

Case Is <= 11 * c
va = 3
car(11) = car(11) + 1

Case Is <= 12 * c
va = 4
car(12) = car(12) + 1

Case Is <= 13 * c
va = 5
car(13) = car(13) + 1

Case Is <= 14 * c
va = 6
car(14) = car(14) + 1

Case Is <= 15 * c
va = 7
car(15) = car(15) + 1
```

```
            Case Is <= 16 * c
            va = 8
            car(16) = car(16) + 1

            Case Is <= 17 * c
            va = 9
            car(17) = car(17) + 1

            Case Else
        End Select

        'Store in AHP worksheet
        Worksheets("AHPCalc").Cells(rw, cl).Value = va

        'Store in Sensitivity worksheet
        Worksheets("Sensitivity.RND").Cells(i + 2, 2 + s).Value = va

        'Increase the row
        rw = rw + 1

        'Do not go above n + 3 rows
        If rw > 3 + n Then
            cnt = cnt + 1
            rw = 5 + cnt
            cl = cl + 1
        End If

    Next i

    'Run the AHP calculation sub
    AHPCalc

    'Store the AHP calculations
    For ahp = 3 To n + 3 - 1

        Worksheets("Sensitivity.RND").Cells(i + ahp, 2 + s).Value =
Worksheets("AHPCalc").Cells(ahp + 1, n + 3).Value
        Worksheets("Sensitivity.RND").Cells(i + ahp + n, 2 + s).Value =
Worksheets("TOPSISCalc").Cells(7, ahp + 1).Value

    Next ahp

    'Activate the TOPSIS sub
    TOPSISCalc
```

```
            'Store the TOPSIS results
            Worksheets("Sensitivity.RND").Cells(i + 2, 2 + s).Value =
Worksheets("TOPSISCalc").Cells(7, n + 4).Value
            Worksheets("Sensitivity.RND").Cells(93 + s, 2).Value =
Worksheets("TOPSISCalc").Cells(7, n + 4).Value

            'Capture the dashboard and store it
            If s = 1 Then

               Worksheets("MCRRM").Range("A2:M22").CopyPicture xlPrinter
               Worksheets("Sensitivity.RND").Paste _
                  Destination:=Worksheets("Sensitivity.RND").Cells(167 + s, 8 + s)

            End If

         Next s

         'Store the number of the separate entries
         For i2 = 1 To 17

            Worksheets("Sensitivity.RND").Cells(119 + i2, 5).Value = car(i2)

         Next i2

      End Sub
```

The mission critical release readiness metrics from running the pairwise comparison sensitivity analysis against the sample case study are shown in Table 14 and the statistical analysis of the results are shown in Table 15. As the statistical analysis indicates, the mean of the 100 runs is 0.918877, the minimum value is 0.8830665, the range is 0.1070358, and the maximum value is 0.990102. The 100 random runs show the influence random pairwise assignments would have on the methodology. By limiting random assignments of pairwise comparisons to a moderate influence, low of 0.8830665

to a high of 0.990102, the methodology demonstrates the importance of correct pairwise comparisons to the software release decision.

Table 14 – Sensitivity Analysis Results

Mission Critical Release Readiness Metric Sensitivity Analysis Results			
0.9901023	0.9303596	0.959848	0.9180587
0.9111727	0.894668	0.8865435	0.9167985
0.889204	0.9293924	0.9123566	0.9488183
0.8875401	0.8937433	0.9274441	0.9014406
0.9478523	0.9142551	0.9442856	0.9030434
0.9500867	0.9173902	0.8960005	0.8970378
0.9091465	0.9354862	0.9232526	0.906713
0.9071067	0.9656863	0.9157838	0.9253478
0.8898708	0.8993347	0.8945422	0.9110429
0.887966	0.9012517	0.8995225	0.9182447
0.9262272	0.8842346	0.891037	0.9323944
0.9208828	0.9390717	0.9037025	0.9153826
0.8830665	0.8995913	0.903492	0.9226818
0.9119383	0.9409368	0.9198534	0.9835958
0.9260248	0.9062865	0.953132	0.8992781
0.9410852	0.9545762	0.9438861	0.9034803
0.9121334	0.9232969	0.9146718	0.9355837
0.9128539	0.9148631	0.9166534	0.9087203
0.9053446	0.910277	0.8961272	0.9188286
0.9141632	0.9265905	0.9430187	0.9450693
0.9544106	0.9199035	0.919082	0.9494938
0.9518933	0.9076719	0.9106458	0.8994398
0.8892658	0.9206037	0.9388658	0.9336378
0.9234938	0.9074235	0.9274353	0.925244
0.9014	0.9190759	0.9083444	0.9186284

Table 15 – Results Analysis

Pairwise Comparison Results Analysis	
Mean	0.9188773
Standard Error	0.002138
Median	0.9162186
Mode	#N/A
Standard Deviation	0.0213802
Sample Variance	0.0004571
Kurtosis	0.7547815
Skewness	0.8013747
Range	0.1070358
Minimum	0.8830665
Maximum	0.9901023
Sum	91.887733
Count	100

Pairwise comparison sensitivity analysis was repeated multiple times using multiple scenarios. Overall, the results indicated the weightings affected the mission critical release readiness metric and that properly selected weightings produced results in-line with customer expectations.

7.6.2.2 Random Pairwise Comparisons And Varying Criterion

The mission critical release readiness methodology was run while randomly generating pairwise comparisons and randomly varying numbers of criterion. The code from above was modified to include randomly varying up to 5 of the 9 case study criterion. The results indicate that as more criterion are randomly varied, the harder it is to have a successful mission critical release readiness metric. Random selections of pairwise comparisons and criterion have a large effect on the mission critical release

readiness metric and their values must be correctly chosen for the methodology to produce meaningful results.

7.6.2.3 Random Pairwise Comparisons And Varying All Criterion

Randomly varying the pairwise comparisons and all criterion produced worst case scenarios during the sensitivity testing. With multiple criterion able to be less than ideal, the mission critical release readiness metric had a mean of 0.036 for 1000 runs and a maximum value of 0.468. The value of the methodology is shown when the pairwise comparisons and criterion are correctly chosen.

7.7 Validation

Validation is ensuring the customer needs and expectations are met by the system. By developing the software release manager's needs through research and interviews, developing those needs into requirements the mission critical release readiness methodology must meet, and verifying the methodology meets the requirements, the customer needs for the mission critical release readiness methodology are validated against the research accomplished. Dynamically testing and applying the mission critical release readiness methodology to real world case studies and producing logic, relevant results provides additional assurance of the methodology's ability to meet the customer's needs against the research accomplished.

7.8 Status

The following sections will be status updates on the mission critical release readiness methodology design space, development, verification and validation, risk, and technical program measures.

7.8.1 Design Space

In section 4.4, the software release methodologies design space was plotted with the candidate software release methodologies plotted against the number of requirements met on the X axis and the number of derived requirements met on the Y axis. The design space is updated to include the mission critical release readiness methodology in Figure 56. The mission critical release readiness methodology is plotted in the upper right of the plot indicating the methodology meets all the requirements and derived requirements.

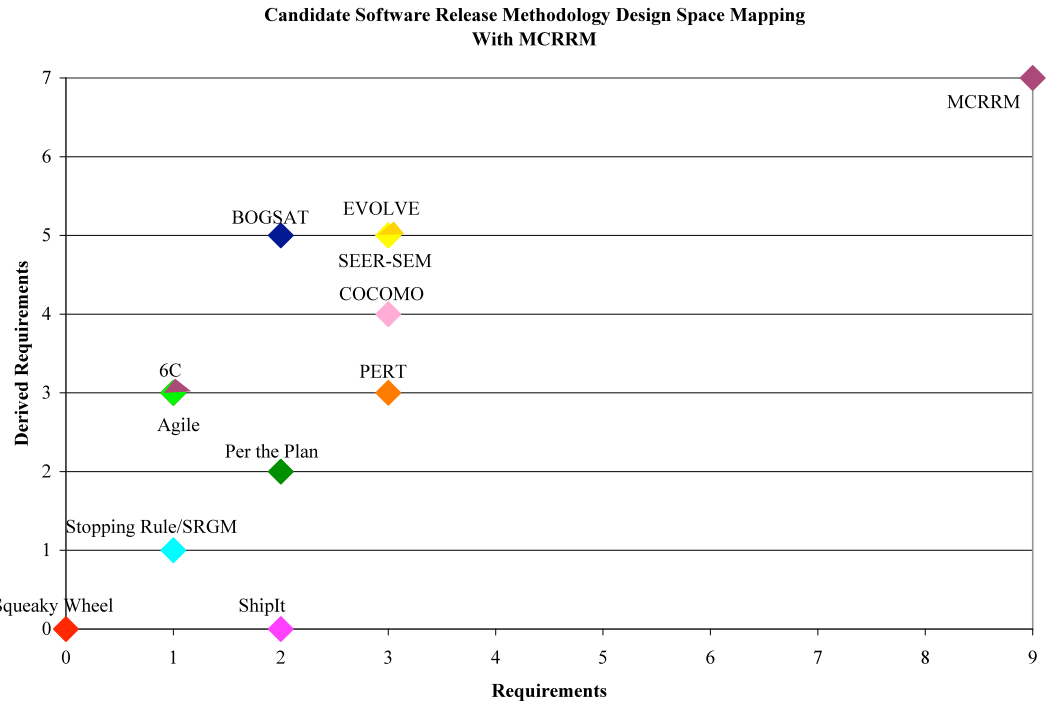

Figure 56 – Design Space Status

7.8.2 Development

The development of the Analytic Hierarchy Process and TOPSIS analytical methods have been documented, tested, and verified. The user interface for the methodology has been documented, tested, and verified. The mission critical release readiness methodology development phase is considered complete.

7.8.3 Verification And Validation

Section 7.5.2 documents the mission critical release readiness methodology requirements and derived requirements verification cross reference matrixes. All

relevant requirements and derived requirements are verified and the verification is considered complete.

The mission critical release readiness methodology was dynamically tested and applied in case studies for validation purposes. Validation of the mission critical release readiness methodology is considered complete.

7.8.4 Risk

Two risks were identified with regard to the mission critical release readiness methodology. The risks and their mitigation plans are shown in Table 16 below for reference. By following the mitigation plans and developing the methodology using common tools and interfaces, using automation, and providing standardized user interfaces for methodology inputs, the likelihood of occurrence for Risk 1 has decreased. During development of the mission critical release readiness methodology, the ease of modification and generic basis behind the methodology facilitated research into other areas to apply the methodology, thus reducing the consequence of Risk 2.

Table 16 – Methodology Risks and Mitigation Plans

RISK	MITIGATION PLAN
1. If the methodology is inefficient at gathering and processing the methodology inputs, then the affect on the methodology would be to use workarounds or add up to 2 additional hours to the processing.	1. Design methodology to use common tools and interfaces. 2. Automate portions of methodology as time allows. 3. Provide standardize data entry forms.
2. If the methodology does not easily apply to a wide variety of software development methodologies and tools, then the methodology will not be widely acceptable and the methodology would be considered almost unusable.	1. Use generic software development labels on data entry forms. 2. Design in methodology customization, where possible.

Although the risks are still valid for the mission critical release readiness methodology, they have been reduced in likelihood and/or consequence and are plotted on the risk matrix for their final status as shown in Figure 57.

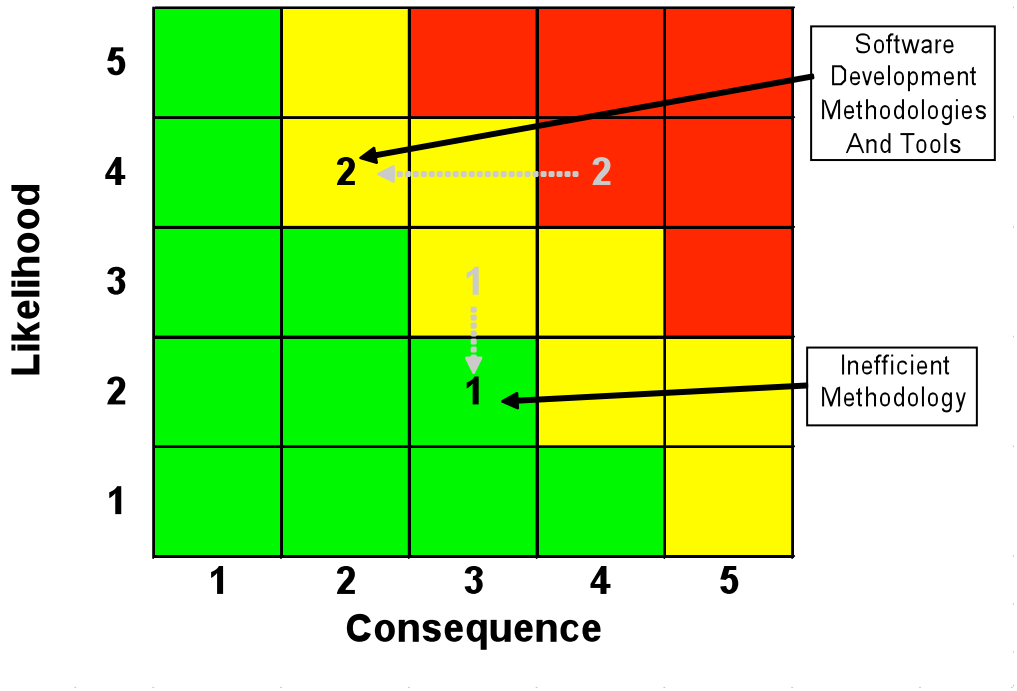

Figure 57 – Updated Risk Items

7.8.5 Technical Performance Measure

One technical performance measure was developed for the mission critical release readiness methodology and it is described below:

TPM1 – The mission critical release readiness methodology shall provide analytically based, release decision support to the user with an objective of less than 1 hour and a not to exceed threshold of 4 hours from the time of initialization, after methodology set-up.

Based on dynamic testing and applying the methodology to use cases demonstrating a near instantaneous calculation upon methodology set-up, the technical performance measure for the mission critical release readiness methodology meets the objective of less than 1 hour.

7.9 Verification And Validation Of Methodology Summary

The mission critical release readiness methodology developed Analytic Hierarchy Process and TOPSIS methodologies are verified via dynamic testing and comparison with commercial analytical tools. A case study application of the mission critical release readiness methodology is presented. The mission critical release readiness methodology's requirements are verified via their verification objectives of either test or inspection. The verification cross reference matrixes for both the requirements and derived requirements and partial validation of the mission critical release readiness methodology are presented. The mission critical release readiness methodology development, verification and validation, risks, and technical performance measure were updated. Summary and conclusions are provided in the next chapter.

Chapter 8

SUMMARY AND CONCLUSIONS

8.1 Process Output

The *Process Output* sub-process is outputting any and all data that characterizes the product or the processes required to develop the product [17]. The output for developing the mission critical release readiness methodology is the process and the conclusions derived from the design, development, verification, and validation of the methodology.

8.2 Present Results

The *Present Results* sub-process of the need identification and problem resolution process summarizes the design, development, verification, and validation of the mission critical release readiness methodology. Additionally, the *Present Results* sub-process summarizes any future work identified during the methodology development.

8.3 Summary

Research was accomplished to determine existing analytical software release decision methodology. The accomplished research found literature lacking for an

existing analytical software release decision methodology, therefore systems engineering processes and analysis were employed to develop a mission critical release readiness methodology.

The process followed was documented and aligned with the systems engineering fundamental process. After researching existing analytical software release decision methodology, user needs were researched, methodology requirements were developed from the user needs, and the requirements were analyzed for missing or derived requirements. With the user needs and methodology requirements developed and documented, brainstorming, analysis, and detailed research were accomplished into possible solutions. A combinational analytical technique was applied to determine if multiple existing methodologies could be combined to meet the needs and requirements. The analytic technique did not reveal a combinational solution and the decision was made to develop the mission critical release readiness methodology.

Additional research and analyses were accomplished regarding the development of the methodology. Development of a metric type methodology ensued. Applying both Analytic Hierarchy Process and TOPSIS analytical methodologies produced an acceptable analytical metric for an analytical software release decision methodology. The analytical basis and user interfaces behind the methodology were developed and the source code for the methodology presented. Throughout research, analysis, and development of the mission critical release readiness methodology, the functional flow block diagrams were developed for the methodology. The final functional flow block diagram for the methodology is shown in Figure 58.

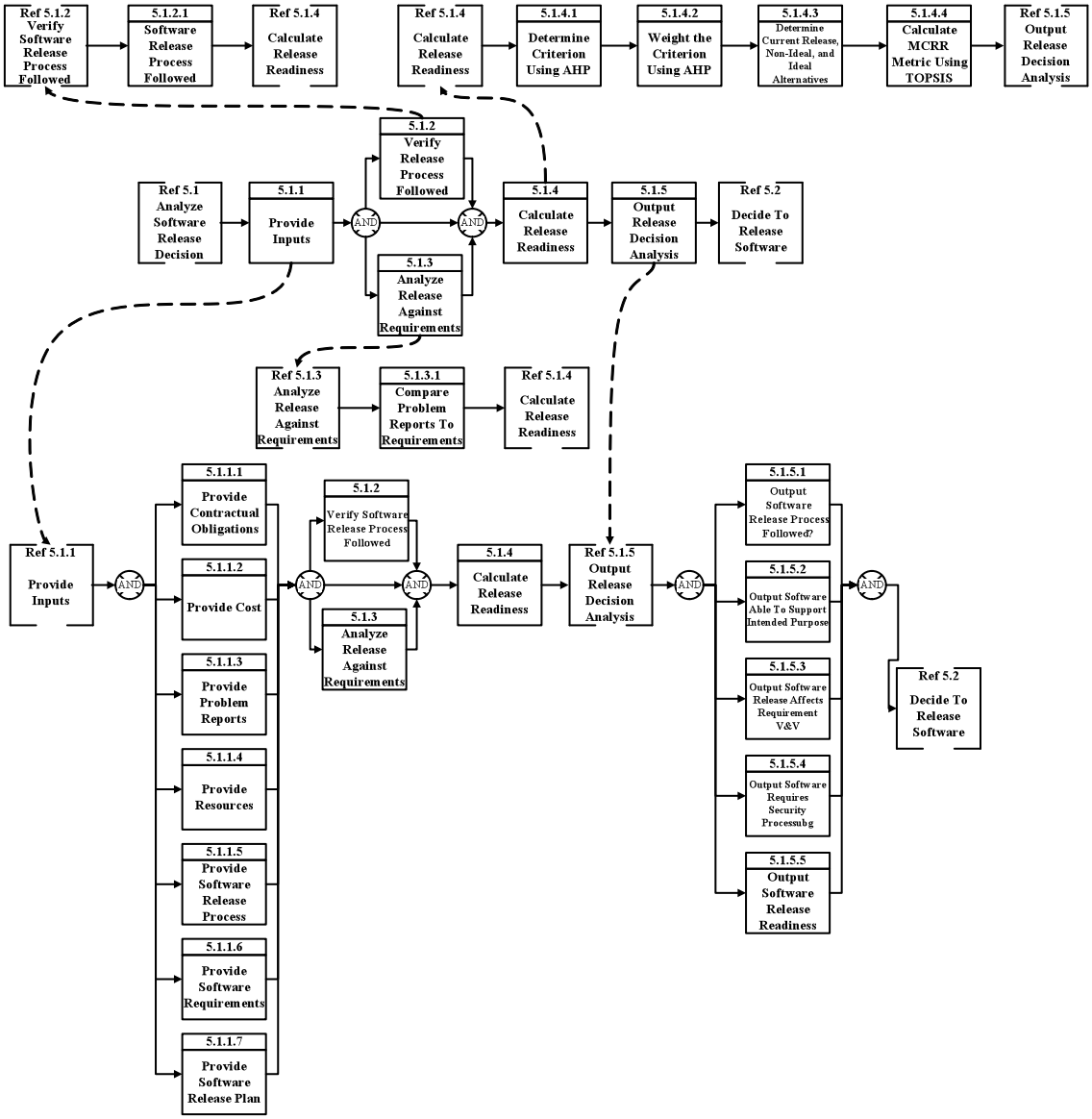

Figure 58 – Complete Methodology

Dynamic testing of the analytical basis of the mission critical release readiness methodology was accomplished and a case study application presented. The results of the dynamic testing and case study application were used during verification of the methodology. A partial validation of the methodology was presented.

The mission critical release readiness methodology dashboard presents the outputs of the methodology in a logical manner, is capable of highlighting cautionary and warning areas for the release, and answers relevant questions required of the software release manager thus providing the software release manager with an analytical decision support methodology to assist in deciding when to release software.

To summarize - in true systems engineering fashion: a need was discovered without a viable solution; a process for responding to the need was developed and aligned with the systems engineering fundamental process; research was accomplished on customer needs and possible solutions; needs were developed into requirements; analysis was accomplished on requirements, derived requirements, and methodology development risks; technical performance measure identified; detailed research and brainstorming were accomplished to define the possible solution design space; a make or "buy" decision was made; the mission critical release readiness methodology was developed; the analytical methodologies behind the mission critical release readiness methodology were verified; methodology testing and case study application ensued; requirements were verified and validated; risks and technical program measures were updated and statused.

8.4 Conclusions

One indication of the usefulness of a methodology is its ease of change. Even with multiple analytical methodologies providing the basis for the mission critical release readiness methodology results, the ease with which the methodology was changed and improved during the development was a relative surprise. Easily updating the

methodology lends the methodology to additional areas of applications outside of the software release decision.

The criticality of software in today's complex systems drives the need for rigor for all phases of the software lifecycle. Research showed a critical need for an analytically based, quantitative mission critical software release decision methodology of direct benefit to the developing complex systems. The developed mission critical release readiness methodology is unlike anything uncovered in a comprehensive literature search or in current known practice. The mission critical release readiness methodology provides a valuable, analytical basis for the mission critical software release decision.

8.5 Future Work

The mission critical release readiness methodology is developed as a proof of concept. Now that the concept has been developed and proofed, the mission critical release readiness methodology should be extended beyond proof of concept. Several ideas for extending the methodology were presented and are collected below:

- A generic methodology was presented with regards to a DoD centric project – showcasing the methodology on commercial specific projects would assist in acceptance of the methodology commercially.
- Automatically query for the inputs to the methodology. This automation would insure the data used was current and assist in meeting the customer need N3 – accurate data. Automation would also assist in meeting customer need N1 – timely access to data support.

- Modeling the release process for use with the methodology, which would assist in both customer needs N1 and N2 – timely running of the methodology and answering whether the software release follows the release process.
- Research into correlating the mission critical release readiness methodology outputs for a particular software release and the problem reports discovered after the software release would make for interesting and appealing future work.
- Applying the analytical basis behind the mission critical release readiness methodology to other decision areas outside of software release is an intriguing area for additional future work.

APPENDIX

A - Mission Critical Release Readiness Methodology Source Code

```
Public Const n As Integer = 9
Public Const m As Integer = 3
```

```
'*********************************************
'*
'*  AHPCalc subroutine calculates the AHP weightings
'*  for criterion based on data from
'*  a spreadsheet.
'*
'*  Author:  Tim Woods          April 2010
'*
'*********************************************

Sub AHPCalc()

    Dim cnt As Integer
    Dim x As Integer
    Dim A(n, n) As Double
    Dim B(n, n) As Double
    Dim RTot(n) As Double
    Dim RNorm(n) As Double
    Dim RNormOld(n) As Double
    Dim RSum As Double
    Dim Ce As Boolean

    '*****************************************
    'Initialize For AHP Calculations

    'Not Close enough, yet, so initialize to False
```

```
Ce = False

'Set counter to -1
cnt = -1

'Clear Weights
For i = 1 To n

   Worksheets("AHPCalc").Cells(3 + i, 3 + n).Value = ""
   RNorm(i) = 0

Next i
'*****************************************


'Iterative process, repeat until close enough
Do While Ce = False

   'Counter = 0 first time through and x places printouts
   cnt = cnt + 1
   x = 3 + cnt * (n + 2)

   'Load the A matrix and clear the B
   For i = 1 To n

      For j = 1 To n

         A(i, j) = Worksheets("AHPCalc").Cells(x + i, 2 + j).Value
         B(i, j) = 0

      Next j

   Next i

   'Clear the Row sum variable
   RSum = 0

   'Loop through, multiplying the A matrix times itself
   For i = 1 To n

      'Keep track of row totals in RTot
      RTot(i) = 0

      For j = 1 To n
```

```
        For k = 1 To n

            'Multiply A times A and store in B
            B(i, j) = B(i, j) + A(i, k) * A(k, j)

        Next k

        'Total rows
        RTot(i) = RTot(i) + B(i, j)

        'Write B matrix for results
        Worksheets("AHPCalc").Cells(2 + x + n + i, 2 + j).Value = B(i, j)

    Next j

    'Record Row totals
    Worksheets("AHPCalc").Cells(2 + x + n + i, 2 + j).Value = RTot(i)

    'Sum up the row totals in RSum
    RSum = RSum + RTot(i)

Next i

'Record the row totals sum
Worksheets("AHPCalc").Cells(2 + x + n + i, 2 + j).Value = RSum

For i = 1 To n

    'Store the old weightings
    RNormOld(i) = RNorm(i)

    'Calculate new weightings and store them
    RNorm(i) = RTot(i) / RSum
    Worksheets("AHPCalc").Cells(2 + x + n + i, 2 + j + 1).Value = RNorm(i)

    'If old and new weightings are within 0.000005, close enough
    If Abs(RNormOld(i) - RNorm(i)) < 0.000005 Then

        Ce = True

    Else

        Ce = False

    End If
```

```
            Next i

            'If looped this many times, cut it off
            If cnt = 2 * n Then

               Ce = True
               Worksheets("AHPCalc").Cells(3 + i, 2 + j).Value = "Did not Converge on
common values after " & cnt & " attempts."

            Else

               'If close enough, write weightings to spreadsheet
               If Ce = True Then

                  For i = 1 To n

                     Worksheets("AHPCalc").Cells(3 + i, 2 + j).Value = RNorm(i)

                  Next i

               End If

            End If

         Loop

         'Record the date the analysis was completed
         TDate = Date
         Worksheets("AHPCalc").Cells(2, 11).Value = TDate

      End Sub
```

```
'*********************************************
'*
'*  TOPSISCalc subroutine calculates the TOPSIS
'*  alternative rankings based using data from
'*  a spreadsheet.
'*
'*  Author:  Tim Woods          April 2010
'*
'*********************************************
```

```
Sub TOPSISCalc()

    Dim cnt As Integer
    Dim x As Integer
    Dim D(m, n) As Double
    Dim V(m, n) As Double
    Dim RTot(m) As Double
    Dim RTot2(m) As Double
    Dim CTot(n) As Double
    Dim CTot2(n) As Double
    Dim Ai(n) As Double
    Dim Ani(n) As Double
    Dim Si(m) As Double
    Dim Sni(m) As Double
    Dim MCRRM(m) As Double
    Dim Wts(n) As Double
    Dim Obj(n) As String

    '*****************************************
    'Initialize For TOPSIS Calculations

    'Clear Variables, Initialize Weights and Objectives
    x = 5

    For i = 1 To m

        RTot(i) = 0
        RTot2(i) = 0
        Si(i) = 0
        Sni(i) = 0
        MCRRM(i) = 0

    Next i


    For j = 1 To n

        CTot(j) = 0
        CTot2(j) = 0
        Wts(j) = Worksheets("TOPSISCalc").Cells(x, 3 + j).Value
        Obj(j) = Worksheets("TOPSISCalc").Cells(x + 1, 3 + j).Value

    Next j

    'Clear the MCCRM Metric and Negative-Ideal and Ideal Alternative Results
```

```
For i = 1 To m

   Worksheets("TOPSISCalc").Cells(x + 1 + i, 4 + n).Value = ""

Next i


'*****************************************


'Load the arrays and total the sum of the squares for the rows and columns
For i = 1 To m

   For j = 1 To n

      D(i, j) = Worksheets("TOPSISCalc").Cells(x + 1 + i, 3 + j).Value
      V(i, j) = 0
      RTot(i) = RTot(i) + D(i, j) ^ 2
      CTot(j) = CTot(j) + D(i, j) ^ 2

   Next j

   RTot2(i) = RTot(i) ^ 0.5

Next i

'Calcualte the normalized matrix
For i = 1 To m

   For j = 1 To n
      CTot2(j) = CTot(j) ^ 0.5
      V(i, j) = Wts(j) * (D(i, j) / CTot2(j))
      Worksheets("TOPSISCalc").Cells(x + 6 + i, 3 + j).Value = V(i, j)

      'Depending upon the Criterion Objective, find the max or min of the
criterion
      If i = m Then

         If V(1, j) >= V(2, j) Then

            MaxV = V(1, j)
            MinV = V(2, j)

         Else
            MaxV = V(2, j)
```

```
            MinV = V(1, j)

        End If

        If V(3, j) >= MaxV Then

            MaxV = V(3, j)

        ElseIf V(3, j) < MinV Then

            MinV = V(3, j)

        End If

        'Load the correct alternative
        If Obj(j) = "Maximize" Then

            Ai(j) = MaxV
            Ani(j) = MinV

        Else 'Objective is to minimize

            Ai(j) = MinV
            Ani(j) = MaxV

        End If

        Worksheets("TOPSISCalc").Cells(x + 8 + i, 3 + j).Value = Ai(j)
        Worksheets("TOPSISCalc").Cells(x + 9 + i, 3 + j).Value = Ani(j)

    End If

  Next j

Next i

For i = 1 To m

  For j = 1 To n

    Si(i) = Si(i) + (V(i, j) - Ai(j)) ^ 2
    Sni(i) = Sni(i) + (V(i, j) - Ani(j)) ^ 2

  Next j
```

```
        'Calculate the alternatives distance from ideal and negative ideal
        Si(i) = Si(i) ^ 0.5
        Sni(i) = Sni(i) ^ 0.5
        Worksheets("TOPSISCalc").Cells(x + 13 + i, 4).Value = Si(i)
        Worksheets("TOPSISCalc").Cells(x + 13 + i, 5).Value = Sni(i)
        MCRRM(i) = Sni(i) / (Si(i) + Sni(i))

        'Write the distance in as the MCRR Metric
        Worksheets("TOPSISCalc").Cells(x + 1 + i, 4 + n).Value = MCRRM(i)

    Next i

    'Record the date the analysis was completed
    TDate = Date
    Worksheets("TOPSISCalc").Cells(2, 12).Value = TDate

End Sub
```

B - Mission Critical Release Readiness Methodology Sensitivity Analysis

Sensitivity analysis was completed by varying each of the criterion and examining the effect the criterion's value had on the final output of the methodology. The case study weightings and inputs were the basis for the analysis and the following figures are the results of the sensitivity analysis.

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	0
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
MCRRM		0.4142

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 59 – Contractual Obligations Metric Warning

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$23,190.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.9796

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Probable

Figure 60 – Cost Dashboard Caution

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$24,540.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.9243

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Probable

Figure 61 – Cost Metric Caution

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$26,790.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
MCRRM		0.8485

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 62 – Cost Metric Warning

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	10
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
MCRRM		0.9901

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Probable

Figure 63 – Problem Reports Dashboard Caution

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	25
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.9381

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 64 – Problem Reports Metric Caution

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	0
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.8234

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 65 – Hardware Metric Warning

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	0
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
MCRRM		0.8234

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 66 – Laboratories Metric Warning

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	0
Software Requirements	12	12
Software Release Plan	2/25/2010	2/25/2010
MCRRM		0.7440

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 67 – Software Release Process Metric Warning

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	11
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.9679

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Probable

Figure 68 – Software Requirements Dashboard Caution

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	10
Software Release Plan	2/25/2010	2/25/2010
	MCRRM	0.9353

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 69 – Software Requirements Metric Caution

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	7
Software Release Plan	2/25/2010	2/25/2010
MCRRM		0.8372

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 70 – Software Requirements Metric Warning

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/26/2010
MCRRM		0.8631

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Probable

Figure 71 – Software Release Plan Dashboard Caution

9/6/2010

MCRRM Dashboard	Current Release	
	Expected Value	Current/ Projected Value
Contractual Obligations	1	1
Cost	$22,740.00	$22,740.00
Problem Reports	8	8
Resources: Hardware	1	1
Resources: Laboratories	1	1
Resources: Personnel	8	8
Software Release Process	1	1
Software Requirements	12	12
Software Release Plan	2/25/2010	2/27/2010
	MCRRM	0.7386

Software Release	
Software Release Affects V&V?	No
Software Release Process Followed?	Yes
Software Requires Security Processing?	No
Software Able To Perform Intended Purpose?	Improbable

Figure 72 – Software Release Plan Metric Warning

REFERENCES

[1] Karim Nice (2001, April 11) How Car Computers Work, Retrieved April 09, 2007 from http://auto.howstuffworks.com/car-computer.htm.

[2] Forte, Rob (2005, May 9) Driving By Wire *Autonet.ca*, Retrieved September 30, 2007, from http://www6.autonet.ca/Parts/Systems/story.cfm?story=/Parts/Systems/2005/05/10/1033945.html.

[3] Leveson, Nancy G., Safeware System Safety and Computers, Addison-Wesley Publishing Company, Inc., New York, New York, ISBN 0-201-11972-2, © 1995.

[4] Schneier, Bruce (2000, March, 15). Software Complexity and Security. *Crypto-Gram Newsletter*, Retrieved September 29, 2007, from http://www.schneier.com/crypto-gram-0003.html.

[5] Lohr, Steve and Markoff, John (2006, March, 15). Windows Is So Slow, but Why? *New York Times* , Retrieved September 29, 2007, from http://www.nytimes.com/2006/03/27/technology/27soft.html?_r=1&oref=slogin#.

[6] Summerville, Ian, Software Engineering, Seventh Edition, Pearson Education Limited, Essex England, ISBN 0-321-21026-3, © 2004

[7] Bays, Michael E., Software Release Methodology, Prentice Hall PTR, Upper Saddle River, New Jersey, ISBN 0-13-636564-7, © 1999.

[8] RTCA/DO-178B. "Software Considerations in Airborne Systems and Equipment Certification", December 1, 1992.

[9] IEEE Software Engineering Coordinating Committee,(SWECC). 2001. Software Engineering Book of Knowledge. http://www.swebok.org/.

[10] Keeney, Ralph L., "Decision Analysis: An Overview" Operations Research, Vol. 30, Iss. 5, pp. 803-838, September 1982.

[11] Goodwin, Paul and Wright George, Decision Analysis for Management Judgment, Third Edition, John Wiley & Sons Ltd., West Sussex, England, ISBN 0-470-86108-8, © 2004.

[12] "severity." The American Heritage® Dictionary of the English Language, Fourth Edition. Houghton Mifflin Company, 2004. Answers.com 19 Aug. 2006. http://www.answers.com/topic/severity

[13] Wikipedia, the Encyclopedia of the Internet, "Definition of Software Release", Wikipedia Foundation, Incorporated, Text is licensed under the GNU Free Documentation License, http://en.wikipedia.org/wiki/Software_Release

[14] "software." (2008). In *Merriam-Webster Online Dictionary*. Retrieved September 15, 2008, from http://www.merriam-webster.com/dictionary/software

[15] "Systems Engineering" (2008) from *INCOSE website*. Retrieved September 15, 2008, from http://www.incose.org/practice/whatissystemseng.aspx

[16] Kockler, Frank R., Withers, Thomas R., Poodiack, James A, Gierman, Michael J., Systems Engineering Management Guide, Defense Systems Management College, 1990.

[17] U.S. Department of Defense (DoD) Systems Management College, Systems Engineering Fundamentals, Defense Acquisition University Press, Fort Belvoir, Virginia 22060-5565, January 2001

[18] Pre-Milestone A and Early-Phase Systems Engineering: A Retrospective Review and Benefits for Future Air Force Acquisition http://www.nap.edu/catalog/12065.html

[19] Blanchard, B.S., Systems Engineering Management. 3rd Ed., Hoboken, New Jersey: John Wiley & Sons, Inc., 2004.

[20] Blanchard, B.S. and Fabryky, W.J., Systems Engineering and Analysis: (2nd ed.). Englewood Cliffs, N.J.: Prentice Hall, 1990.

[21] Fishman, Charles, "They Write the Right Stuff" Issue 06| Dec 1996/Jan 1997 | Page 95 Retrieved September 16, 2008, From: http://www.fastcompany.com/magazine/06/writestuff.html?page=0%2C0

[22] Robat,Cornelis, editor, "The History of Software" retrieved October 17, 2008, from http://www.thocp.net/software/software_reference/introduction_to_software_history.htm

[23] Babcock, Daniel L., Managing Engineering and Technology: an introduction to management for engineers, Prentice Hall, Englewood Cliffs, New Jersey, ISBN 0-13-552233-1, © 1991.

[24] Cohen, Bob, and Lykins, Howard, "Modeling and Systems Engineering Working Group Report", *INCOSE INSIGHT*, Volume 4 Issue 2, July 2001.

[25] Beck, Kent, Extreme Programming Explained: Embrace Change, Addison-Wesley, Boston, Massachusetts, ISBN 0-321-27865-8, © 2005.

[26] Tyson, Jeff, "How BIOS Works". Retrieved October 21, 2008, from http://computer.howstuffworks.com/bios1.htm

[27] Newby, Timothy J., Stepich, Donald A., Lehman, James D., and Russel, James D., Educational Technology for Teaching and Learning – 3rd ed. Pearson Merrill Prentice Hall, Upper Saddle River, New Jersey, ISBN 0-13-046714-6, © 2006.

[28] Franklin, Curt and Coustan, Dave, How Operating Systems Work. Retrieved October 21, 2008, from http://computer.howstuffworks.com/operating-system.htm.

[29] Pfleeger, Shari Lawrence, Hatton, Les, and Howell, Charles C., Solid Software, Pearson Merrill Prentice Hall, Upper Saddle River, New Jersey, ISBN 0-13-091298-0, © 2002.

[30] Haskins, Cecilia, ed., "INCOSE Systems Engineering Handbook Version 3", © 2006.

[31] Brief History of Systems Engineering, Retrieved November 2, 2008, from http://www.incose.org/mediarelations/briefhistory.aspx.

[32] Frey, Dan, Clausing, Don, and Hale, Pat, ESD.33 --Systems Engineering Session #1, "Course Introduction: What is Systems Engineering?", Retrieved November 2, 2008, from http://ocw.mit.edu/NR/rdonlyres/Engineering-Systems-Division/ESD-33Summer2004/1E55A228-F8A6-4217-AA9D-BBC21E600310/0/s1_cors_intro_v7.pdf.

[33] "Genesis of INCOSE", Retrieved November 2, 2008, from http://www.incose.org/about/genesis.aspx.

[34] "A Consensus of the INCOSE Fellows", Retrieved November 2, 2008. from http://www.incose.org/practice/fellowsconsensus.aspx.

[35] "About INCOSE", Retrieved November 7, 2009, from http://www.incose.org/about/index.aspx.

[36] Boehm, Barry, A Spiral Model of Software Development and Enhancement, IEEE Computer, vol.21, #5, May 1988, pp 61-72.

[37] U.S. Department of Defense (DoD) Instruction 5000.2, "Operation of the Defense Acquisition System," May 12, 2003.

[38] McCarthy, Jim and McCarthy, Michele, Dynamics of Software Development, 2006 Edition, Microsoft Press, Redmond Washington, Library of Congress Control Number 2006924464, © 2006.

[39] Free Software Foundation, Inc., "The Free Software Definition", Retrieved on November 5, 2008 from http://www.gnu.org/philosophy/free-sw.html.

[40] Wasson, Charles S, System Analysis, Design and Development, John Wiley & sons, Inc., Hoboken, New Jersey, ISBN-13 978-0-471-39333-7, © 2006.

[41] Chrissis, Mary Beth, Konrad, Mike, and Shrum, Sandy, CMMI: guidelines for process integration and product improvement, 2nd edition, Pearson Education, Inc., Boston Massachusetts, ISBN – 0-321-27967-0, © 2007.

[42] Hermann, Brian G. and Russel, Jim, "Are You Ready to Deliver? To Ship? To Test?", *STSC – CrossTalk, The Journal of Defense Software Engineering,* August 1998

[43] Kerkhoff, Wimm, "Software Release Management", Retrieved on November 16, 2008 from http://www.nyetwork.org/wiki/srm.pdf.

[44] Bach, James, "Reframing Requirements Analysis", Retrieved on November 30, 2008, from http://www.satisfice.com/articles/reframing_requirements.pdf.

[45] Hooks, Ivy, "Writing Good Requirements (A Requirements Working Group Information Report)", Proceedings of the *Third International Symposium of the NCOSE* - Volume 2, 1993.

[46] Roedler, Gary, J. and Jones, Cheryl, "Technical Measurement, A Collaborative Project of PSM, INCOSE, and Industry", Retrieved on November 30, 2008, from http://www.incose.org/ProductsPubs/pdf/TechMeasurementGuide_2005-1227.pdf

[47] Ruhe, Günther and Saliu, Moshood Omolade, "The Art and Science of Software Release Planning", November/December 2005 *IEEE SOFTWARE*, Pages 47-53.

[48] Prince, Frank A., C and the Box, A paradigm Parable, Pfeiffer & Company, San Diego, California, ISBN: Hardcover 0-88390-364-4, © 1993.

[49] "Idiom: Squeaky wheel gets the grease", UsingEnglish.com, Retrieved on December 3, 2008, from http://www.usingenglish.com/reference/idioms/squeaky+wheel+gets+the+grease.html

[50] Neves, Sue and Strauss, Jack, "Survival Guide for Truly Schedule-Driven Development Programs", *AT&L*: July-August 2008, Pages 21-23.

[51] Siok, Michael F., Whittaker, Clinton J., and Tian, Jeff, "Exposing Software Field Failures", *STSC – CrossTalk, The Journal of Defense Software Engineering,* November 2006.

[52] Bai,Do S., Yun,Won Y., "Optimal Software Release Policy with Random Life Cycle", *IEEE Transactions on Reliability*, vol R-39, 1990 June, pp 167-170.

[53] Gokhale, Swapne S., "Optimal Software Release Time Incorporating Fault Correction", Proceedings of the *28th Annual NASA Goddard Software Engineering Workshop (SEW'03)*, 2003.

[54] Fischman, Lee, McRitchie, Karen, and Galorath, Danial D., "Inside SEER-SEM", *STSC – CrossTalk, The Journal of Defense Software Engineering,* April 2005.

[55] "USC COCOMO II 2000 Software Reference Manual" Retrieved on January 6, 2009 from http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_manual2000.0.pdf.

[56] "PERT" Retrieved on January 11, 2009 from http://www.netmba.com/operations/project/pert.

[57] Render, Barry, Stair, Ralph M. Jr., and Hanna, Michael E., Quantitative Analysis for Management (Eighth ed.). Upper Saddle River, New Jersey, ISBN: 0-13-066952-0 Prentice Hall, 2003.

[58] Platt, David S., Why Software Sucks…and what you can do about it, Upper Saddle River, New Jersey, ISBN: 0-321-46675-6, Addison-Wesley, 2007.

[59] "COCOMO II", Retrieved January 18, 2009 from http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html.

[60] Greer, D. and Ruhe, G., "Software release planning: an evolutionary and iterative approach", *Information and Software Technology*, Volume 46, Issue 4, 15 March 2004, Pages 243-253.

[61] Bhawnani, P., Far, B. H., and Ruhe, G., "Explorative Study to Provide Decision Support for Software Release Decisions", Proceedings of the *21st IEEE International Conference on Software Maintenance* (September 25 - 30, 2005), ICSM, IEEE Computer Society, Washington, DC.

[62] Highsmith, J., "What is Agile Software Development", *STSC – CrossTalk, The Journal of Defense Software Engineering,* October 2002.

[63] Paulk, Mark C., "Agile Methodologies and Process Discipline", *STSC – CrossTalk, The Journal of Defense Software Engineering,* October 2002.

[64] Jansen, Slinger and Brinkkemper, Sjaak, "Ten Misconceptions about Product Software Release Management explained using Update Cost/Value Functions", Proceedings of the *International Workshop on Software Product Management* (IWSPM'06), September, 2006

[65] U.S. Department of Defense (DoD), "*Risk Management Guide for DoD Acquisition*", Version 1.0, OUSD (AT&L) Defense Systems/Systems Engineering/Enterprise Development, August, 2006.

[66] Wysocki, Robert K., Effective Software Project Management. Indianapolis, Indiana, ISBN -13: 978-0-7645-9636-0 Wiley Publishing, Inc., 2006.

[67] Lyu, Michael R., Handbook of Software Reliability Engineering, McGraw-Hill Companies, New York, NY, 1996, ISBN 0-07-039400-8.

[68] "Is Your Software Ready for Release?," *IEEE Software*, vol. 6, no. 4, pp. 100, 102, 108, July/Aug. 1989, doi:10.1109/MS.1989.10039

[69] Satapathy, Piyush Ranjan "Evaluation of Software Release Readiness Metric [0,1] across the software development life cycle", Retrieved on March 5, 2010 from http://www.cs.ucr.edu/~piyush/SoftwareProj_Report.pdf.

[70] Boehm, Barry, Valerdi, Ricardo, Lane, Jo Anne, and Brown, A. Winsor, "COCOMO Suite Methodology and Evolution", *STSC – CrossTalk, The Journal of Defense Software Engineering,* April 2005.

[71] Hong, G. Y., Xie, M., Zhao, M., and Wohlin, Claes, "Interval Estimation in Software Reliability Analysis", Proceedings *4th International Applied Statistics in Industry Conference*, pages 105-112, Kansas City, Missouri, USA, 1997.

[72] Mahmoodzadeh, S., Sharhrabi, J., Pariazar, M., and Zaeri, M. S., "Project Selection by Using Fuzzy AHP and TOPSIS Technique", *International Journal of Human and Social Sciences,* Volume 1, Number 3, pages 135 – 140, 2007.

[73] Saaty, Thomas L., "Relative Measurement and its Generalization in Decision Making: Why Pairwise Comparisons are Central in Mathematics for the Measurement of Intangible Factors - The Analytic Hierarchy/Network Process".

RACSAM (Review of the Royal Spanish Academy of Sciences, Series A, Mathematics), Volume 102, Number 2, pages 251–318, 2008.

[74] PERT, Program Evaluation Research Task, Phase I Summary Report, vol. 7, Special projects office, Bureau of Ordinance, U.S. Department of the Navy, Washington, D.C., 1958.

[75] Thuesen, G. J. and Fabryky, W.J., Engineering Economy: (8th ed.). Englewood Cliffs, N.J.: Prentice Hall, 1993.

[76] Devi, Kavita, Yadav, Shiv P., and Kumar, Surendra, "Extension of Fuzzy TOPSIS Method Based on Vague Sets", *International Journal of Computational Cognition*, vol.7, no.4, pages 58-62, December 2009.

[77] Wagner, J. F., "An Implementation of the Analytic Hierarchy Process (AHP) on a Large Scale Integrated Launch Vehicle Avionics Systems Engineering Architecture Trade Study", Proceedings of the *Ninth Annual Symposium of the International Council on Systems Engineering*, Volumes 1 & 2, June 6–11, 1999, Brighton, England.

[78] Daniels, J., Werner, P.W., and Bahill, A.T., "Quantitative Methods for Tradeoff Analyses" , *Systems Engineering The Journal of the International Council on Systems Engineering,* Volume 4 Number 3, pages 190–212, 2001.

[79] Saaty, T.L., "Decision Making with the Analytic Hierarchy Process", *International. Journal Services Sciences*, Volume 1, Number 1, pages 83-98, 2008.

[80] Lintner, Thomas M., Smith, Steven D., Smurthwaite, Scott, "The Aerospace Performance Factor: Utilization Of The Analytical Hierarchy Process To Develop A Balanced Performance And Safety Indicator Of The National Airspace System For The Federal Aviation Administration", Proceedings of the *10th International Symposium on the Analytic Hierarchy/Network Process,* July 29 – August 1, 2009, Pittsburgh, Pennsylvania.

[81] Watkins, John C., Ghan, L. Scott, "AHP Version 5.1 User's Manual", Prepared for Division of Systems Research, U.S. Nuclear Regulatory Commission, DOE Contract No. DE-AC07-76ID01570, 1990, by Idaho National Engineering Laboratory, Idaho Falls, Idaho.

[82] Carlen, Eric A., "Additional notes on the power method for finding eigenvectors", MATH 2605 Course Outline and Calendar, Retrieved on April 16, 2010 from http://people.math.gatech.edu/~carlen/2605S04/Power.pdf.

[83] Bluebit, Online Matrix Calculator, online at http://www.bluebit.gr/matrix-calculator/default.aspx.

[84] "TOPSIS", Retrieved on April 17, 2010 from http://stat-design.com/topsis-sdi.php.

[85] Jones, E.L., "The SPRAE Framework for Teaching Software Testing in the Undergraduate Curriculum", Proceedings of *ADMI 2000*, June 1-4, 2000.

[86] IEEE Standard Glossary of Software Engineering Terminology, IEEE Standard 610.12-1990, 1990.

[87] McCabe, Thomas J., "A Software Complexity Measure", IEEE Transactions on Software Engineering, Vol. 2, pp 308-320, 1976

[88] Costea, Adrian, "On Measuring Software Complexity", Journal of Applied Quantitative Methods, Volume 2, Issue 1, 2007.

[89] Clark, M., Brennan, D., Salesky, B., and Urmson, C., Measuring Software Complexity to Target Risky Modules in Autonomous Vehicle Systems, AUVSI Unmanned Systems North America, June 2008.

[90] Estep, Donald, "Calculus-Based Approaches to Sensitivity Analysis", Retrieved on September 5, 2010, from http://www.math.colostate.edu/~estep/research/talks/nrel_lecture_1.pdf.

[91] Forsberg, Kevin and Mooz, Harold, The Relationship of System Engineering to the Project Cycle, National Council On Systems Engineering (NCOSE) Conference, Chattanooga, TN, 21–23 October 1991.