



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

JOINT APPLIED PROJECT

Software Independent Verification
& Validation (SIV&V) Simplified

By: **Reffela Davidson,
Ashley Mathis,
David Patterson, and
Alexis P. von Spakovsky
December 2006**

Advisors: **Bill Craig
Brad Naegle**

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 2006	3. REPORT TYPE AND DATES COVERED Joint Applied Project	
4. TITLE AND SUBTITLE: Software Independent Verification and Validation (SIV&V) Simplified			5. FUNDING NUMBERS	
6. AUTHOR(S) Reffela Davidson, Ashley Mathis, David Patterson, Alexis von Spakovsky				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this report are those of the author(s) and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT SIV&V has been in existence for some 40 years, and many people still know little about its existence. Software IV&V certifies the quality of the software and independently validates and verifies that it meets or exceeds the customer's expectations. Independent V&V for component or element software development activities encompasses the following: 1) review and thorough evaluations of the software development, 2) review and comment on software documentation, 3) participation in all software requirements and design reviews, and 4) participation in software integration and testing for each software build. This thesis will explore and explain the benefits and rationale for Software Independent Verification and Validation. It will identify SIV&V processes that are used to support acquisition weapon systems. "SIV&V Simplified" will translate, into understandable terms, why SIV&V is considered "Cheap Insurance" and why it is needed. Additionally, this thesis serves as a tutorial, providing suggested policy and guidance, suggested software Computer-Aided Software Engineering (CASE) tools, criteria, and lessons learned for implementing a successful SIV&V program.				
14. SUBJECT TERMS Software, Independent, Verification, Validation, Policy, Guidance, SIV&V, Computer Aided Software Engineering, CASE tools, Acquisition, Documentation, Test and Evaluation, Development, Requirements, Design Reviews, Integration, Benefits, Processes, Weapons Systems, Cheap Insurance, Criteria, Lessons Learned			15. NUMBER OF PAGES 115	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**SOFTWARE INDEPENDENT VERIFICATION AND VALIDATION (SIV&V)
SIMPLIFIED**

Reffela Davidson, GS-12, United States Army IMMC, GMD
Ashley Mathis, NH-IV, Missile Defense Agency, THAAD
David Patterson, NH-III, United States Army Aviation, Space and Missiles
Alexis P. von Spakovsky, NH-IV, Missile Defense Agency, GMD

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN PROGRAM MANAGEMENT

from the

**NAVAL POSTGRADUATE SCHOOL
December 2006**

Authors:

Reffela Davidson

Ashley Mathis

David Patterson

Alexis P. von Spakovsky

Approved by:

Dr. Bill Craig, Lead Advisor

LTC (ret) Brad Naegle, Support Advisor

Robert N. Beck, Dean
Graduate School of Business and Public Policy

THIS PAGE INTENTIONALLY LEFT BLANK

SOFTWARE INDEPENDENT VERIFICATION AND VALIDATION (SIV&V) SIMPLIFIED

ABSTRACT

Software Independent Verification and Validation (SIV&V) has been in existence for some 40 years, and many people still know little about its existence. Software IV&V certifies the quality of the software and independently validates and verifies that it meets or exceeds the customer's requirements and expectations. Independent V&V for component or element software development activities encompasses the following: 1) review and thorough evaluations of the software development, 2) review and comment on software documentation, 3) participation in all software requirements and design reviews, and 4) participation in software integration and testing for each software build. This thesis will explore and explain the benefits and rationale for Software Independent Verification and Validation. It will identify SIV&V processes that are used to support acquisition weapon systems. "SIV&V Simplified" will translate, into understandable terms, why SIV&V is considered "Cheap Insurance" and why it is needed. Additionally, this thesis serves as a tutorial, providing suggested policy and guidance, suggested software Computer-Aided Software Engineering (CASE) tools, criteria, and lessons learned for implementing a successful SIV&V program.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	PURPOSE.....	1
B.	BACKGROUND.....	1
C.	RESEARCH QUESTIONS.....	2
	1. Primary Research Question.....	2
	2. Secondary Research Questions.....	2
D.	SCOPE OF THESIS.....	2
E.	METHODOLOGY.....	2
	1. Data Collection Methodology.....	2
	2. Data Analysis Methodology.....	3
	3. Conclusions and Recommendations Methodology.....	3
F.	ORGANIZATION.....	3
G.	BENEFITS OF STUDY.....	4
II.	THE SIV&V BACKGROUND.....	5
A.	INTRODUCTION.....	5
B.	KEY DEFINITIONS.....	5
C.	SIV&V BACKGROUND.....	6
	1. Historical.....	6
	2. SIV&V Policy/Guidance.....	10
	3. The Role of SIV&V.....	13
	4. SIV&V Challenges.....	14
III.	THE SIV&V GUIDE.....	17
A.	SIV&V SIZING AND TYPES.....	17
	1. Introduction.....	17
	2. Survey of SIV&V Metrics.....	17
	3. SIV&V Staffing Levels.....	19
B.	THE BENEFITS OF SIV&V.....	21
	1. Introduction.....	21
	2. Costs Associated with Discovered Errors.....	22
	3. Return On Investment (ROI).....	25
	a. <i>Average Source Line of Code (SLOC) Development Cost.....</i>	<i>27</i>
	b. <i>Analysis of SIV&V Software Trouble Reports Corrected and Implemented by the Developer.....</i>	<i>27</i>
	c. <i>Multipliers for Relative Cost to Correct Defects.....</i>	<i>28</i>
	d. <i>Return On Investment Calculation.....</i>	<i>31</i>
	e. <i>Development Schedule Reduction.....</i>	<i>33</i>
	f. <i>More Measurable Results of SIV&V.....</i>	<i>34</i>
	g. <i>Examples With and Without SIV&V.....</i>	<i>37</i>
	h. <i>DoD SIV&V Examples:.....</i>	<i>39</i>
	i. <i>Commercial Examples: No SIV&V Agent Utilized ...</i>	<i>40</i>

C.	APPLYING SIV&V IN THE CYCLE PHASE	41
1.	Introduction.....	41
2.	Concept Definition Phase	42
3.	Requirements Phase	45
4.	Design Phase	48
a.	<i>Architecture Design</i>	49
b.	<i>Detailed Design</i>	51
5.	Code and Development Testing Phase.....	54
6.	Hardware and Software Integration Phase	56
7.	Formal Qualification Phase.....	59
8.	Operational Readiness Phase	61
9.	Operations and Maintenance Phase	63
D.	SIV&V SUPPORTING CASE TOOLS.....	64
1.	Introduction (I-Logix.com; Telelogics; Spector; Rational; Zambrana)	64
2.	Requirements.....	66
3.	Design.....	66
4.	Code.....	66
5.	Tracking Database	67
6.	Rate Monotonic Analysis (RMA).....	67
7.	Security Assessment (Klocwork; Ghosh; Gilliam; Laliberte).....	68
E.	IMPACTS OF ACQUISITION REFORM	69
F.	KEYS TO SUCCESSFUL SIV&V.....	71
IV.	SUMMARY, RECOMMENDATIONS, AND CONCLUSIONS	77
A.	SUMMARY	77
B.	CONCLUSIONS	77
C.	RECOMMENDATIONS	78
D.	ANSWERS TO RESEARCH QUESTIONS	80
1.	Primary Research Question.....	80
2.	Secondary Research Questions.....	80
E.	RECOMMENDATIONS FOR FURTHER STUDY	83
	APPENDIX: INTEGRATED SYSTEM DIAGRAM (ISD).....	85
	LIST OF REFERENCES.....	87
	INITIAL DISTRIBUTION LIST	93

LIST OF FIGURES

Figure 1.	Technical Support And Program Size (Missile Defense Agency 15) ..	21
Figure 2.	Cost To Fix Defect Versus Life Cycle Phase (Boehm 1981 40)	23
Figure 3.	Average Cost of Discovering Errors (Lewis 1992 279)	31
Figure 4.	Ariane 5 Crash (Knutson; Missile Defense Agency 24)	37
Figure 5.	NASA SIV&V Policy (Missile Defense Agency 23)	38
Figure 6.	THAAD Electronics Box (U.S. Army 28)	40
Figure 7.	Notional SIV&V Activities (Walters)	42
Figure 8.	SIV&V Battle Rhythm Concept Development Phase (Lewis 1998)	45
Figure 9.	SIV&V Battle Rhythm Requirements Phase (Lewis 1998).....	48
Figure 10.	SIV&V Battle Rhythm Architectural Design Phase (Lewis 1998).....	51
Figure 11.	SIV&V Battle Rhythm Detailed Design Phase (Lewis 1998).....	53
Figure 12.	SIV&V Battle Rhythm Code and Development Testing Phase (Lewis 1998).....	56
Figure 13.	SIV&V Battle Rhythm Hardware and Software Integration Phase (Lewis 1998).....	58
Figure 14.	SIV&V Battle Rhythm Formal Qualification Phase (Lewis 1998)	61
Figure 15.	SIV&V Battle Rhythm Operational Readiness Phase (Lewis 1998) ...	62
Figure 16.	SIV&V Battle Rhythm Operations and Maintenance Phase (Lewis 1998)	64

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Major Commercial Software Failures (Department of Air Force 2-7)	9
Table 2.	SIV&V Policy And Guidance.....	10
Table 3.	SIV&V Metrics	18
Table 4.	System B	28
Table 5.	System D.....	28
Table 6.	Study Multipliers	29
Table 7.	Recommended SIV&V Tool Suite	69

THIS PAGE INTENTIONALLY LEFT BLANK

ACRONYMS AND ABBREVIATIONS

AAMI	Association for the Advancement of Medical Instrumentation
ADR	Architectural Design Review
AF	Air Force
AFI	Air Force Instruction
AFLCP	Air Force Logistics Command Pamphlet
AFSC	Air Force Systems Command
AFSMC	Air Force Space and Missile Systems Center
AMC	Army Material Command
AMRDEC	Aviation and Missile Research Development and Engineering Center
ANS	American Nuclear Society
ANSI	American National Standards Institute
AT&T	American Telephone and Telegraph
CASE	Computer-Aided Software Engineering
CDR	Critical Design Review
CM	Carnegie Mellon
CM	Configuration Management
CMM	Capability Maturity Model
COBOL	Common Business Oriented Language
COTS	Commercial-Off-The-Shelf
COU	Concept of Operations and Utilization
CR	Contractor Report
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSU	Computer Software Unit
DAU	Defense Acquisition University
DBDD	Database Design Document
DDR	Detailed Design Review

Dem/Val	Demonstration/Validation
DoD	Department of Defense
DT&E	Developmental Test and Evaluation
ECP	Engineering Change Proposal
ESD	Electronic Systems Division
FAA	Federal Aviation Administration
FCA	Functional Configuration Audit
FEU	Functional Equivalent Unit
FHWA	Federal Highway Administration
FIPS	Federal Information Processing Standard
FMEA	Failure Mode Effects Analysis
FORTTRAN	Formula Translator
FQT	Formal Qualification Test
FRR	Flight Readiness Review
FUE	Field Unit Equipped
GB	Guidebook
GMD	Ground-Based Midcourse Defense
GOTS	Government-Off-The-Shelf
GSAM	General Service Administration Acquisition Manual
GTE	General Telephone and Electronics
GUI	Graphical User Interfaces
HSI	Hardware and Software Integration
HSI	Human Systems Integration
HTP	Hardware Test Plan
HWCI	Hardware Configuration Item
I/O	Input/Output
I ² V ²	Independent Integrated Verification and Validation
IBM	International Business Machines
ICD	Interface Control Document
IDD	Interface Design Document

IEEE	Institute for Electrical and Electronic Engineers
IOC	Initial Operational Capability
IPT	Integrated Product Team
IRS	Interface Requirements Specification
ISD	Integrated System Diagram
ITD	Integrated Test Description
ITP	Integrated Test Plan
ITT	International Telegraph and Telephone
KSLOC	Thousand Source Line of Code
LOE	Level of Effort
MOE	Measure of Effectiveness
MOP	Measure of Performance
MTT	Methods, Tools, and Techniques
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
NBS	National Bureau of Standards
NIST	National Institute of Standards and Technology
NIST	National Institute of Standards and Technology
NPD	NASA Policy Directive
NSWCDD	Naval Surface Warfare Center - Dahlgren Division
NUREG	Nuclear Regulatory Commission
O&M	Operation and Maintenance
OCD	Operational Concept Document
OIG	Office of Inspector General
ORD	Operational Requirements Document
ORR	Operational Readiness Review
OT&E	Operational Test and Evaluation
P3I	Pre-Planned Product Improvement
PAM	Pamphlet
PCA	Physical Configuration Audit

PD	Program Director
PDR	Preliminary Design Review
PDSS	Post-Deployment Software Support
PIDS	Prime Item Development Specification
PM	Program Manager
PMO	Project Management Office
PO	Program or Project Office
POC	Point of Contact
RAM	Reliability, Availability, Maintainability
RDECOM	Research Development and Engineering Command
RMA	Rate Monotonic Analysis
ROI	Return on Investment
SAT	Security Assessment Tool
SCR	Software Change Requests
SDD	Software Design Document
SDF	Software Development Folder
SDO	Software Development Organization
SDP	Software Development Plan
SDR	Systems Design Review
SED	Software Engineering Directorate
SEE	Software Engineering Environment
SEES	Software Engineering Evaluation System
SEI	Software Engineering Institute
SI	Software Item
SIV&V	Software Independent Verification & Validation
SLOC	Source Line of Code
SLP	System Level Procedure
SME	Subject Matter Expert
SOW	Scope of Work
SOW	Statement of Work

SPR	Software Problem Report
SPS	Software Product Specification
SRR	System Requirements Review
SRS	Software Requirements Specification
SS	System Specification
SSDD	System Segment Design Document
SSR	Software Specification Review
STD	Software Test Description
STD	Standard
STP	Software Test Plan
STR	Software Trouble Reports
SU	Software Unit
SW	Software
TEMP	Test and Evaluation Master Plan
THAAD	Theater High Altitude Area Defense
TRD	Technical Requirements Document
TRR	Test Readiness Review
TRW	Thompson Ramo Wooldridge
UML	Unified Modeling Language
US	United States
USS	United States Ship
V&V	Verified and Validated
WBS	Work Breakdown Structure

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

The authors would like to thank our thesis advisors, Dr. Bill Craig and LTC (ret) Brad Naegle. Their support, time, and advice helped guide the authors and make this project less of an ordeal, given our demanding school, work, and personal lives. Additionally, we would like to thank the U.S. Army's Acquisition Education Training and Experience Board for giving us this opportunity to further our educations and careers. And to the faculty and staff of the Naval Postgraduate School, thank you for your dedication to teaching us and helping us expand our horizons. Finally, we would like to thank our families, our bosses, and our co-workers for their endless patience and understanding as we struggled to balance our responsibilities between home, work, and school. Without their support, none of our efforts over the last two years would have been successful.

I (Alexis von Spakovsky) would like to personally thank my co-workers Alan Bollers and Hap Terrell for encouraging me to apply for this graduate program and providing me constructive edits for my application package; my bosses James Johnson, Tom Lancaster, and Gina Gilbertson for allowing me the time I needed to go back to school; my co-workers, Wayne Gardner, Mark Douglas, Don Smith, Kim Miller, Allan Bollers, and Rick Wegman for taking up the slack at work; and most especially my wife Ida and my children, Victor, Leos, Emilia, and Josef, for sacrificing fun time and vacations, and at times putting up with their extremely tired and grumpy father. Thank you!!!

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

The purpose of this thesis is to explain the benefits and rationale of Software Independent Verification and Validation (SIV&V) to Program Management Offices (PMO) and others. Additionally, this thesis serves as a tutorial, providing policy and guidance documentation, as well as, suggested software Computer-Aided Software Engineering (CASE) tools, criteria, and lessons learned to implement a successful SIV&V program.

B. BACKGROUND

The costliest, most complex and critical component of almost every weapons system employed by the Armed Forces of the United States is the software. The software is that key component of a weapon system that allows it to perform its functions. For the war-fighters to be able to successfully carry out their missions, the software must be able to perform in its operational environment(s) per requirements. Failure to do so can lead to mission failure and catastrophic consequences for the war-fighters and their equipment. An excellent quote that illustrates this point was made by LTG Robert H. Ludwig, and simply states the “Fly-by-Wire F16C ... without software,” is nothing more than “... a 15-million dollar lawn dart!” (Department of Air Force 1-14).

When the National Security of the United States and the lives of its men and women in uniform are at stake, this country, especially members of the Defense community, must strive to provide the very best engineered software products that money and a disciplined engineering process can produce. SIV&V becomes a critical tool in verifying and validating highly complex software that helps assure the war-fighters’ weapon systems will be operational and perform as required when they are needed most.

C. RESEARCH QUESTIONS

1. Primary Research Question

What are the benefits of and rationale for PMOs and others for using SIV&V?

2. Secondary Research Questions

What software CASE tools are available for software V&V?

What key things should be done or considered when conducting SIV&V?

What are the SIV&V process steps?

How has acquisition reform affected SIV&V?

What lessons have been learned from past programs that have utilized SIV&V?

D. SCOPE OF THESIS

This thesis serves as a tutorial on SIV&V for the Program Manager and others engaged in the acquisition of systems for the United States Armed Forces. As such its scope is limited to providing a brief history of SIV&V, highlighting some policy and guidance documentation, discussing acquisition reform, SIV&Vs importance to the overall success of producing systems that are both operationally effective and suitable, and providing an understandable practical methodology for conducting SIV&V. Several real world examples are used to convey the benefits of employing SIV&V, the pitfalls of not using SIV&V, and providing lessons learned that can benefit programs that apply this methodology.

E. METHODOLOGY

1. Data Collection Methodology

The data collection methodology for this thesis consisted of extensive research of available SIV&V materials from books, the internet and online library sources, government and other policies, regulations, standards and handbooks. Additionally, the more than 11 years of SIV&V experience of one of the authors, Mr. Ashley Mathis, was instrumental in bringing this information together in an understandable, concise and practical form.

2. Data Analysis Methodology

All of the data was analyzed from the perspective of real world experience with SIV&V. The data analysis was conducted to determine what did and did not work, how to apply or not to apply the method, why to apply the method, and when to apply the method. A critical eye was applied to looking for practical, straightforward ways in which to apply SIV&V methods and lessons learned that could be easily understood and applied to any acquisition program.

3. Conclusions and Recommendations Methodology

This thesis will aid PMOs in making informed decisions when faced with political and budgetary realities. Thus, the conclusions and recommendations of this thesis stemmed from a careful analysis of the data through the prism of real world experiences. The objective was to provide a clear understanding of the need for and the benefits of a properly applied SIV&V methodology. Armed with this information, PMOs will have understanding of a powerful tool that is critical to the successful acquisition of today's complex software centric systems.

F. ORGANIZATION

This thesis consists of four chapters.

Chapter I – Introduction - This chapter establishes the purpose, background, the research questions, the scope, the methodology, organization, and benefits of this thesis.

Chapter II – SIV&V Background - This chapter introduces the material, providing a description, definitions, and a history of SIV&V's evolution in conjunction with the technological progress of computers and software. Also addressed are some guidelines and policies, the role, and the future challenges facing SIV&V.

Chapter III – The SIV&V Guide - This chapter addresses sizing and types of SIV&V agents, metrics, and staffing levels. Additionally, the benefits of SIV&V are addressed, using real world examples to illustrate the differences between programs that used SIV&V and ones that did not. Finally, the chapter discusses how to apply SIV&V throughout the life cycle of a program, the CASE tools that

can be used to manage the effort, the impacts of acquisition reform, and key strategies for conducting SIV&V successfully.

Chapter IV – Summary, Recommendations, and Conclusions - This chapter summarizes the answers to the research questions from chapter I, and recommends an area for further study.

Appendix – The appendix provides a chart showing the Integrated Systems Diagram process that the Software Engineering Directorate uses. This is followed by the List of References.

G. BENEFITS OF STUDY

This study will benefit PMOs and others who are acquiring systems in support of the American men and women in uniform. The PMOs will gain an advantage by having a better understanding of the benefits of utilizing an effectively tailored SIV&V process that will allow them to produce highly reliable software while reducing the life cycle costs of their programs. Further, the PMOs will have a better understanding of the process steps, methodologies, and tools that are critical to successfully applying SIV&V to their programs. Finally, PMOs will have a readily available reference that specifies various SIV&V policies and guidelines and where they can be found. Ultimately, it is the warfighter who will reap the greatest benefits through fielding of highly reliable systems that are operationally effective and suitable.

II. THE SIV&V BACKGROUND

A. INTRODUCTION

“Modern aerospace and defense systems incorporate increasingly sophisticated information processing and control systems” (Department of Air Force U-3). Given the ever increasing complexity of these systems, policy makers recognizing the necessity for and the benefits of SIV&V, have instituted policies, regulations, and guidance making SIV&V an integral part of modern systems acquisition. One such regulation is Department of Defense (DoD) Directive 5000.1, which identifies Software Independent Verification and Validation (SIV&V) as providing the Program or Project Office (PO) with an independent assessment of the software. Thus, it becomes the responsibility of the SIV&V agent to ensure systems operate and continue to demonstrate a high-level of reliability throughout their life cycles. This document is a synopsis of the overarching processes and justification for the SIV&V effort, and is used as the top-level guideline for all SIV&V activities. The SIV&V Background section will discuss the nature and rationale for the SIV&V environment.

B. KEY DEFINITIONS

Independent: The use of a team that is separate from the development and development management/oversight teams to perform V&V.

Software Verification: “Are we building the right thing right?” - Confirmation by examination and provision of objective evidence that specified requirements fulfill the output of a particular phase of development and meet all the input requirements for that phase.

- Verifies software architecture design based upon software requirements, Software Development Plan (SDP), and the Software Design Document (SDD)
- Analyzes interfaces, data flow, exception handling, timing budgets, memory allocations
- Compares code to design and development requirements

Software Validation: “Are we building the right thing?” - Establish objective evidence that all software requirements are correctly implemented, complete and are traceable to system requirements. Software validation is a design verification function and includes all of the verification and testing activities conducted throughout the software life cycle.

- Ensures that all software requirements are qualified (certified) through analysis, inspection, demonstration, or test
- SIV&V assesses software capability to meet specified design and performance requirements

Software Independent Verification and Validation (SIV&V): An independent risk reduction processes applied to operational software to ensure the prepared code is properly built to specifications and adequately tested for deployment in a delivered operational system.

C. SIV&V BACKGROUND

1. Historical

To truly appreciate and comprehend how and why SIV&V developed, and the critical part that it plays in the development of today’s complex systems, a brief review of the history of computers and software development is needed.

The first electronic computer was developed in the 1940s (Rombach 52). These early computers did not separate the hardware from the software, as they were built to perform one task or solve one problem, and thus were switched or hard-wired to perform certain tasks (Robot 5-7). By the 1950s, computers had progressed to single-user operating systems, supported high-level programming languages such as COBOL (Common Business Oriented Language) (Codasyl committee 1960), and FORTRAN (FORmula TRANslator) (IBM 1952) (Robot 11), and multiple computer applications (Rombach 52). In the 1960s computers became more powerful, supported multi-user operating systems, a variety of more intricate applications, and were used extensively to solve ever more complex problems (Rombach 52). By 1968, computers and software had evolved to a level of complexity where the term “The Software Crisis” was

applied for the first time at a NATO conference in Garmisch –Partenkirchen (Rombach 53), to describe the growing problem with software quality and reliability. The ensuing decades of the 70s, 80s, and 90s have only added to the problem. As computers have become more capable and the software more complex, engineers have utilized these technical advances to solve ever more sophisticated and intricate problems. As a result, we have seen software grow from a relatively small contribution to system development cost of 20% in the 1950s, to 80% of system cost in the 1980s, to nearly 95%-100% of system cost today (Reiss 397-398). This change in cost between hardware and software can be explained as follows. The cost of the hardware for the large, early computers was very expensive; typically, very few were built, and those that were occupied the space of several rooms. As a result of technological advances and the utilization of mass production techniques, computer hardware became not only more capable and smaller, but also much cheaper. Further, in the past, large software programs consisted of thousands of lines of code. In contrast, today's systems consist of millions of lines of code. Software engineers have found that as lines of code increased arithmetically, the cost and complexity of the software tended to increase exponentially. New methodologies and techniques were required to develop and manage the ever-increasing size of these software systems (Reiss 397-398).

From its infancy in the 1940s to today, "...software development has evolved from small tasks involving a few people to enormously large tasks involving many people" (Tran 1-2). Similarly V&V has changed from an "...informal process performed by the software engineer himself..." to a "...highly formalized..." "...separate activity..." conducted by "...organizations independent of the software developer..." and "...practiced over the entire software life cycle" (Tran 1-2).

Given the historical context of computing and software, it is not surprising that V&V and IV&V of software developed at a much later date, not because it was planned, but rather as a necessity for dealing with increasing software

complexity and the resulting lack of quality and reliability. In the earliest days of computers and software, there was a significant "...lack of discipline in the software development process ..." (Persons 5-7). This early "V&V" process, if you could call it such, was nothing more than the programmer debugging their code (Shridhar 2-3). As software began to mature in the late 1950s, the U.S. Department of Defense (DoD) began to notice that as their systems started incorporating more software, three issues kept making repeat appearances, namely, budget overruns, schedule delays, and technical problems (Food For Thought 1-2). These recurring issues necessitated that a more formal method of developing and managing software be established. One of the very first uses of V&V was in the DoD on "...the Atlas Missile Program in the late 1950s" (Food For Thought 1-2).

In 1962, the Air Force learned an expensive lesson with the loss of an Atlas booster and its Mariner payload because of a simple software error (Shridhar 2-3). This failure resulted in the Air Force mandating that all future mission critical software would require independent verification (Shridhar 2-3). It was this requirement that served as the catalyst for what we know today as IV&V (Shridhar 2-3).

In the 1970s, as software began to migrate to the commercial sector, many of these software companies began to experience the same issues that had plagued government programs a decade earlier (Food for Thought 2-3). It was during this same time frame that "... the U.S. Army sponsored the first significant such IV&V program for the Safeguard Anti-Ballistic Missile System. This program pushed IV&V from a fledgling stage to being a mature systems and software engineering discipline.... It was from this effort that IV&V became well known within the Department of Defense and aerospace communities as an accepted method of ensuring better quality, performance, and reliability of critical systems.... By the mid- to late 1970s, IV&V was rapidly becoming popular and in

some cases was required by the military services, especially for systems that has a high cost of failure and hence were able to justify the small added cost of IV&V” (Lewis 1992 xxiii).

Even as the Defense and Aerospace communities were embracing an evolving SIV&V as a viable methodology for handling software quality and reliability, their counterparts in the commercial sector continued to experience quality, budget, schedule, and technical problems on an alarming scale as late as the 1980s and 1990s (Food for Thought 2-3). Table 1 (Department of Air Force 2-7) shows some examples of major commercial software failures.

Table 1. Major Commercial Software Failures (Department of Air Force 2-7)

YEAR	PROJECT	RESULTS
1980s	International Telegraph & Telephone (ITT) – 4 Switching Systems	40,000 Function Point System, \$500M Lost, Cancelled
1987	California Department of Motor Vehicles, Automated Vehicle/Drivers License System	3 (5,000 Function Point Size) Switches, \$30M Lost, Cancelled
1989	State of Washington – Automated Social Services Caseworker System	7 Years to Build, Failed to Meet User Needs, \$20M Lost, Cancelled
1992	American Airlines – Flight Booking System	\$165M Lost, Cancelled

Over the past 60 years, we have witnessed the birth, growth, and development of computers and software. As the technology matured, and became intertwined throughout our modern infrastructure, its very complexity required the software engineering discipline and its process to grow and mature

in an effort to control and manage this software. A natural outgrowth of this evolution was the need for SIV&V; a mindset, processes, and a set of tools developed to provide software engineers and PMs the capability to consistently produce reliable, quality software.

The authors expect that computers and software technology will continue to progress and mature, becoming ever more complex and sophisticated. The future successes of software programs will become ever more dependent on the SIV&V process, and as such, will continue to force the evolutionary advancement of SIV&V so that it remains a viable and effective methodology in the never-ending battle to tame and manage the expanding complex nature of software.

2. SIV&V Policy/Guidance

Many agencies both government and commercial have devised policies, regulations, and standards that address SIV&V. Table 2, listing policies and guidelines, is not all-inclusive but serves as a handy reference that can assist and guide the reader in implementing SIV&V within their own projects and organizations.

Table 2. SIV&V Policy And Guidance

(Table continues on following pages)

Policy/Regulation/Standard/Other	Agency	Website/Comments
AFSC/AFLCP 800-5 "Software Independent Verification and Validation"	US Department of the Air Force (AF)	http://segoldmine.ppi-int.com/menu_guides.htm
AFSMC Regulation 800-26 "Independent Verification and Validation"	US Department of the Air Force, Space and Missile Systems Center (AFMC)	http://www.fas.org/spp/military/docops/smc/ivv26.htm
AFI 16-1001 "Verification, Validation, and Accreditation"	US Department of the Air Force	http://www.e-publishing.af.mil/pubfiles/af/16/afi16-1001/afi16-1001.pdf
ESD-TR-326 "Software Acquisition Management Guidebook: Validation and Certification"	US Department of the Air Force, Electronic Systems Division, Hanscomb AFB	http://stinet.dtic.mil/oai/oai?&verb=getRecord&metadataPrefix=html&identifier=ADA053039

IEEE 1012-2004 "IEEE Standard for Software Verification and Validation"	Institute for Electrical and Electronics Engineers	http://www.ieee.org/web/standards/home/index.html
ANSI/IEEE 1074-2006 "IEEE Standard for Developing a Software Project Life Cycle Process"	American National Standards Institute/Institute for Electrical and Electronics Engineers	http://www.ieee.org/web/standards/home/index.html http://webstore.ansi.org/ansidocstore/product.asp?sku=1074%2D1997
IEEE 1059-1993 "IEEE Guide for Software Verification and Validation Plans"	Institute for Electrical and Electronics Engineers	http://www.ieee.org/web/standards/home/index.html
NPD 8730.4 NASA Policy Directive	National Aeronautics and Space Administration	www.ivv.nasa.gov/foremployees/policyplans.php Established NASA Policy for Independent Verification and Validation. Replaced by 2820.1C.
NPD 2820.1C NASA Policy Directive	National Aeronautics and Space Administration	www.ivv.nasa.gov/foremployees/policyplans.php NASA Independent Verification and Validation Policy
"Independent Verification and Validation Implementation Plan 2003-2008"	National Aeronautics and Space Administration	www.ivv.nasa.gov/foremployees/policyplans.php
"Independent Verification and Validation Implementation Plan 2005-2010"	National Aeronautics and Space Administration	www.ivv.nasa.gov/foremployees/policyplans.php
NASA OIG IG-03-011 "Independent Verification and Validation of Software"	National Aeronautics and Space Administration	http://oig.nasa.gov/audits/reports/FY03/pdfs/ig-03-011.pdf NASA Office of Inspector General Audit Report
NASA-STD-8739.8 "Software Assurance Standard"	National Aeronautics and Space Administration	http://www.hq.nasa.gov/office/codeq/doctree/87398.pdf
NASA-STD-8719.13A "Software Safety"	National Aeronautics and Space Administration	http://satc.gsfc.nasa.gov/assure/nss8719_13.html
"Program Plan for the NASA Software Independent Verification and Validation Program" Rev 1 May 04	National Aeronautics and Space Administration	www.ivv.nasa.gov/foremployees/policyplans.php

NASA-GB-002-95 "Formal Methods Specification and Verification Guidebook for Software and Computer Systems"	National Aeronautics and Space Administration	http://www.fing.edu.uy/inco/grupos/mf/TPPSF/Bibliografia/fmguid1.pdf
ANSI/ANS 10.4-1987;R1998 "Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry"	American National Standards Institute/ American Nuclear Society	http://www.ans.org/store/vi-240150
BSR/AAMI SW76-200x "Software Verification and Validation for High-risk Medical Devices"	Association for the Advancement of Medical Instrumentation	http://www.nssn.org/search/DetailResults.aspx?docid=41766&selnode
FHWA Handbook V1.2 "Verification, Validation, and Evaluation of Expert Systems: An FHWA Handbook"	Federal Highway Administration	http://www.tfhr.gov/advanc/vve/cover.htm http://www.tfhr.gov/advanc/vve/toc.htm
FIPSPUB 101 "Guideline for Life Cycle Validation, Verification, and Testing of Computer Software"	US Department of Commerce, National Bureau of Standards	Federal Information Processing Standards (FIPS) http://www.itl.nist.gov/fipspubs/withdraw.htm Withdrawn and replaced by industry standards. Can still be bought from http://www.nssn.org/search/DetailResults.aspx?docid=263282&selnode
FIPSPUB 132 "Guideline for Software Verification and Validation Plans"	US Department of Commerce, National Bureau of Standards	Federal Information Processing Standards (FIPS) http://www.itl.nist.gov/fipspubs/withdraw.htm Withdrawn and replaced by industry standards (IEEE 1012).
NBS Special Publication 500-93 "Software Validation, Verification, and Testing Technique and Tool Reference Guide"	US Department of Commerce, National Bureau of Standards	http://library.nist.gov/uhtbin/cgiirsi/VweuQgyAsh/NIST/117380059/123
NIST Special Publication 500-234 "Reference Information for the Software Verification and Validation Process"	US Department of Commerce, National Institute of Standards and Technology	http://hisa.nist.gov/HHRFdata/Artifacts/ITLdoc/234/val-proc.html
NIST Special Publication 500-165 "Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards"	US Department of Commerce, National Institute of Standards and Technology	http://library.nist.gov/uhtbin/cgiirsi/VweuQgyAsh/NIST/117380059/123

NIST Special Publication 500-223 "A Framework for the Development and Assurance of High Integrity Software"	US Department of Commerce, National Institute of Standards and Technology	http://hissa.nist.gov/publications/sp223/
NUREG/CR-6316 Volumes 1-8 "Guidelines for the Verification and Validation of Expert System Software and Conventional Software"	US Nuclear Regulatory Commission	http://www.osti.gov/energycitations/searchresults.jsp?Author=Mirsky,+S.M. This report is an excellent source of information.
PAM 5-11 Verification, Validation, and Accreditation of Army Models and Simulations"	US Department of the Army	http://www.army.mil/usapa/epubs/pdf/p5_11.pdf
SEI-CM-13-1.1 "Introduction to Software Verification and Validation Module"	Carnegie Mellon University, Software Engineering Institute	http://www.sei.cmu.edu/publications/documents/cms/cm.013.html
NASA Briefing "Software Independent Verification and Validation"	National Aeronautics and Space Administration	A briefing on how to conduct SIV&V. http://ses.gsfc.nasa.gov/ses_data_2001/010307_Bruner_IVV.ppt
NASA SLP IVV 09-1 Rev. I Effective March 2006	National Aeronautics and Space Administration	This is a System Level Procedure (SLP). http://ims.ivv.nasa.gov/sharedfiles/documents/IVV_09-1.doc
NASA PD-ED-1228 "Independent Verification and Validation of Embedded Software"	National Aeronautics and Space Administration	NASA preferred reliability practice. http://klabs.org/DEI/References/design_guidelines/design_series/1228msfc.pdf

3. The Role of SIV&V

The primary role of SIV&V is to provide the Program Director (PD) or Management with an independent assessment capability to guarantee:

- Software is developed and tested to ensure high confidence in the system and component capabilities
- Software is mature and dependable
- Technical issues and trends are identified in a timely manner for Management focus
- Risk reduction of Program, Element or Component failures due to operational or simulation software defect(s)

These activities include software design evaluation, code inspections, code assessments, and independent test reviews. The overall objective of software V&V is to insure that the product is free from failures and meets its user's requirements and expectations.

Broadly speaking, it can be stated that some level of SIV&V should be used on the following classes of systems (Defense Acquisition University 386):

- Real-time critical software systems that must work every time they are used
- Programs having critical outputs that cannot be verified on every run
- Programs having a high cost of failure in terms of human life, national security or money
- Software for which the cost of error detection through operational use or testing exceeds the cost of employing SIV&V
- Software for which the cost of support is expected to exceed the cost of using SIV&V

Thus, the description of SIV&V as “cheap insurance” becomes patently obvious, particularly when one considers the consequences of failure of critical software, especially when lives are at stake.

4. SIV&V Challenges

In the previous three sections we have discussed the emerging need of SIV&V from a historical perspective, the policy and guidelines developed by both the government and commercial entities, and the primary role and uses of SIV&V. Given that the reader accepts the premise of the last three sections, and understands the need for and uses of SIV&V, the challenge lies in how to effectively apply a SIV&V strategy to existing acquisitions as well as incorporating SIV&V as an integral part of new acquisitions.

The first part of this challenge is to educate management on the what, how, and necessity of SIV&V, to get the buy in required for implementation.

The second part of this challenge is to find the financial resources required to carry out SIV&V and to allocate them appropriately.

The third part of this challenge is to educate the workforce in the SIV&V process; that is to understand when it makes sense, where it makes sense, how to tailor it, how to implement it, and how to utilize it to improve their software processes and products.

The fourth part of this challenge is to understand that SIV&V is not the proverbial “silver bullet” that will fix poor software development processes, poor requirements development and control, poor configuration management, poor systems/software engineering, poor quality control, or poor documentation. It is however, a method of “cheap insurance,” that will allow a good software process to be even better by finding problems before software gets deployed to the user.

The final part of this challenge lies in understanding the nature of the SIV&V process as one that is constantly evolving and developing in an effort to keep pace with the rapidly changing world of software development. This will necessitate that those utilizing SIV&V and SIV&V agents stay current with developments in the software and SIV&V fields and continually educate their workforces so that they remain both effective and relevant.

THIS PAGE INTENTIONALLY LEFT BLANK

III. THE SIV&V GUIDE

A. SIV&V SIZING AND TYPES

1. Introduction

Software IV&V is indeed an important aspect of developing quality software. In many programs, scarce amounts of funds are allocated to SIV&V efforts. Thus, each endeavor should be tailored so that it corresponds with the level of criticality of the software development effort. This section will discuss and describe the different types and sizing metrics of SIV&V agents

2. Survey of SIV&V Metrics

Several key SIV&V metrics were established for comparing the size, responsibility, cost, and performance of both the respective SIV&V teams and the Developer across various systems. The cumulative data is presented in Table 3. Names of actual programs and associated SIV&V agents were changed to protect the confidential nature of the information.

Table 3. SIV&V Metrics

SIV&V Metric	System A	System B	System C	System D
Percent of SIV&V Budget to Development Budget	6.6%	4.9%	4%	8%
Percent of SIV&V LOE to Development LOE	4.3%	8.2%	5.7%	15%
Ratio of SIV&V LOE to # of KSLOC	1:72	1:44	N/A	1:13
Ratio of SIV&V LOE to # of SW Requirements	1:355	1:525	N/A	1:350
Development Cost per SLOC	\$150	\$275	N/A	\$125
Average Cost to Fix Error (Before SW Release)	Est. \$0.5K	N/A	N/A	\$2.5 per SLOC
Average Cost to Fix Defect (After SW Release)	Est. \$1K	Est. \$20.6K	N/A	\$12.5 per SLOC
Ratio of SIV&V LOE Cost to Developer LOE Cost	1:1.54	1:1.7	N/A	1:1.89

(N/A – Not Available)

The metrics indicate several trends common across all of the surveyed efforts. Items of interest include:

- All of the SIV&V efforts are budgeted less than 10% of the software development budget.
- The majority of the SIV&V efforts have staffing levels less than 10% of that of the Developer.
- On average, each SIV&V analyst is responsible for 410 software requirements and/or 43 KSLOC.
- Average SIV&V labor rates are approximately 59% of the Developer rates.

As indicated above most SIV&V groups are not properly resourced and operate in a degraded mode. However, even in a degraded mode of operation or service they always provide benefits to the program and development effort. From the research provided above, each individual on the SIV&V team is responsible for approximately 300 - 550 requirements and 10 – 98 thousand SLOC. These metrics can be used for future cost and resource estimation purposes.

3. SIV&V Staffing Levels

There are four generally accepted types (or levels) of SIV&V as documented by Lewis (Lewis 1992 12-13):

- Full, In-Phase SIV&V – A comprehensive program spanning requirements phase through post-deployment support. SIV&V costs for this type are typically 8-17% of the total program software development budget. The System D SIV&V program at Software Engineering Directorate (SED) is an example of the complexity of a Full, In-Phase SIV&V.
- Partial SIV&V – A less comprehensive program than Full SIV&V, Partial SIV&V begins during the design or early coding phases and has limited involvement with requirements analysis. SIV&V costs are typically 6-13% of the total software development budget.
- Endgame SIV&V – This level of SIV&V is focused primarily on the test and integration phase. SIV&V costs are typically 2-7% of the total software development budget. The GMD programs at SED (Sea Based X-band Radar, Embedded Test, and Ground Based Interceptor) are examples of Endgame SIV&V support.
- Audit-level SIV&V – This is often referred to as “over-the-shoulder” SIV&V and is a minimal effort. Often, a tiger team is used to determine the adequacy of the software development process.

Additionally, Lewis asserts, "Most past SIV&V programs have cost between 2% and 18% of the software development cost. The higher-cost

programs invariably had hardware and/or software development costs like simulations and tools embedded in them. For beginning SIV&V cost estimates, try to begin between 8% and 10% of the software development cost estimate. SIV&V programs that are funded at less than 4% to 5% of the development cost will have to begin to delete some routinely performed tasks.” (Lewis 1992 280)

During the course of this study, it was found that SIV&V levels were typically that of the Endgame variety, and the costs for the SIV&V programs varied from 2% to 10% of the weapon system’s overall software budget, with an average of 5.3%. After analyzing the levels of SIV&V and the activities conducted on these programs, and given the relatively low percentage of program funding provided for SIV&V, most of the programs had to delete some routine tasks in order to provide the most positive impact to the program with limited funding.

For a secondary reference, Figure 1 below provides a general recommendation for the percentage of technical support in relation to the size of the program as depicted by the AMC School of Engineering and Logistics. Most SIV&V programs are usually at the lower end of this range regardless of the program size.

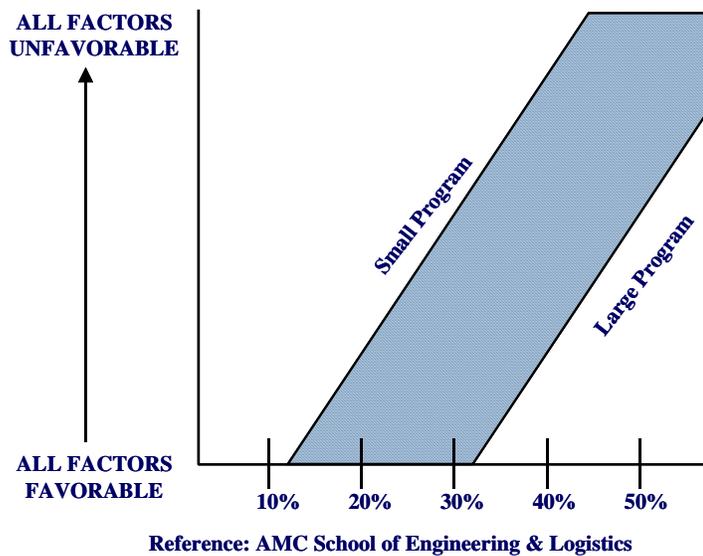


Figure 1. Technical Support And Program Size (Missile Defense Agency 15)

B. THE BENEFITS OF SIV&V

1. Introduction

Without question, the majority of weapon systems have become increasingly software centric with each generation of system that is fielded. The allocation of budget to software engineering has proportionally grown as the systems have expanded and matured. A key program management technique employed by many project offices that has successfully reduced software risks and increased confidence in performance attainment is Software Independent Verification and Validation (SIV&V).

A significant portion of the activities at the U.S. Army, Research, Development, and Engineering Command (RDECOM), Aviation and Missile Research, Development and Engineering Center (AMRDEC), Software Engineering Directorate (SED) at Redstone Arsenal, Alabama, are dedicated to the execution of SIV&V programs for a cadre of important weapon systems. These programs vary in budget, size, and complexity. The weapon systems represent the full range of phases in the development cycle and fielding.

The purpose of Section B is the following: 1) Provide data that supports the value of implementing a SIV&V program on a software intensive system; 2) Provide recommendations based on SIV&V success stories and lessons learned to help improve the acquisition of software intensive systems; 3) Recommend an approach for increasing the probability of successful software program development.

2. Costs Associated with Discovered Errors

A fundamental tenant of SIV&V is that the earlier in the software life cycle in which an error is detected, the cheaper it is to fix. This is usually expressed in terms of the software phase in which the error is detected versus the cost to fix the error. The same relationship can be expressed in relative terms such as, a requirements error discovered during operation costs approximately 200 times more to fix than finding the error during requirements definition (Boehm 1981). SIV&V studies use this approach to quantify SIV&V savings. The most universally utilized relationship is from Barry Boehm's classic textbook, *Software Engineering Economics* (Boehm 1981). His study showed that the relationship is logarithmic. This is shown in the Figure 2:

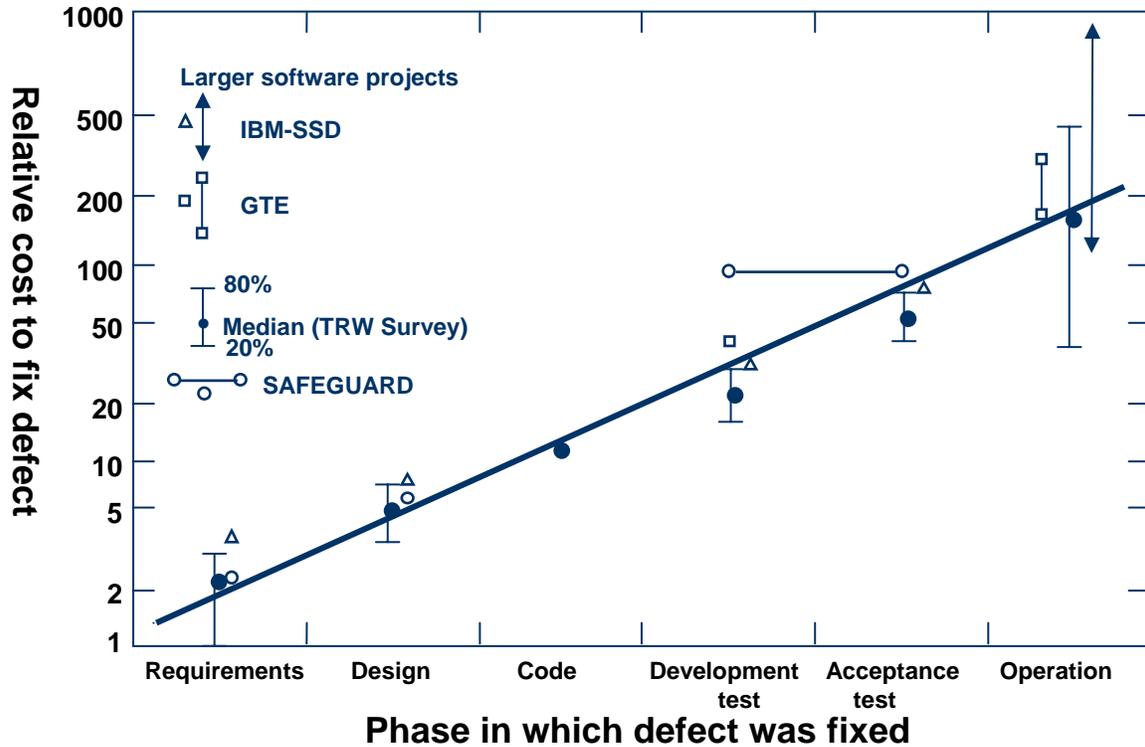


Figure 2. Cost To Fix Defect Versus Life Cycle Phase (Boehm 1981 40)

Since the purpose of this section is to calculate such costs, a mathematical representation of the figure may be useful. By using the midpoint of the six phases projected to the given line and numbered 1 to 6, a table of values can be generated and an exponential equation calculated. The approximate equation is:

$$y = .717e^{.921x}$$

A table lookup approach will also work. This is presented in the Table 6, "Study Multipliers," later in this document.

Experience shows that not all errors are created equal. This is why all Software Problem Report (SPR) systems utilize a priority scheme. The most commonly used scale is 1 to 5, where 1 indicates a safety-critical item or prevents mission accomplishment, and 5, which typically indicates an insignificant documentation issue.

How does this relate to the value of SIV&V? Simply stated, an SIV&V team that routinely finds higher priority defects is more valuable than a team that routinely finds insignificant documentation errors. A simplistic approach to handle this would be to include only Priority 1, 2, and 3 SPRs in the value-added calculation. Although this approach simplifies the calculation, it ignores two key points. First, a single Priority 1 SPR might be much more valuable than several Priority 3 SPRs combined. Second, finding many lower level SPRs is still of value. Again, not all defects and SPRs are created equal. In addition, not all SIV&V findings are associated exclusively with an SPR. Therefore, basing the calculation solely on Priority 1, 2, and 3 SPRs misses some value-added features of an SIV&V team.

A more realistic approach to determine SIV&V value is to draw upon a logarithmic weighting calculation similar to the relative cost calculation approach indicated previously. A simple three “bin” approach for mission critical findings, major findings, and findings scaled at 100 units, 10 units, and 1 unit, respectively, gives adequate granularity with a simple calculation. It is then straightforward to calculate total units divided by level-of-effort (LOE) to obtain value per person during a period of time. Summary metrics for multiple groups and trend graphs can then be easily generated.

Please note the importance of the LOE being included. For example, a team of five people finding ten problems is more valuable than a team of fifty people finding the same ten problems. Calculating the value over a constant period is necessary to show trends and process improvement. The use of the generic term “unit” is also intentional. By eliminating discussion of costs up front, the true issue of productivity (units) is highlighted, and is not clouded by dollars.

The following example from System D at SED illustrates this method. A team of 30 finds 3 mission critical findings, 51 major findings, and 14 basic findings. The calculation of value-added is simply:

$$(3*100 + 51*10 + 14*1) / 30 = 824 \text{ units} / 30 \text{ LOE} = 27.5 \text{ units per person}$$

Assigning findings into three “bins” is similar to assigning priorities to SPRs. Mission critical findings would include such findings as: mission saving, safety critical, system-of-systems improvement, and DoD/Army-level impact. Major findings could include: test failures prevented/detected, major process problems, significant cost savings, production improvement, and significant interoperability issues.

The list for findings is more general and flexible to allow the full scope of SIV&V value-added to be included. This is allowed in this approach since each finding is only worth one unit.

SPRs identified by the SIV&V team would be included as follows:

- Category 1 SPR = Mission Critical Findings
- Category 2 SPR = Major Findings
- Category 3 SPR = Findings

In summary, the appeal of this approach is multifaceted. It is simple to use. It can be used on small or large programs. It can be used on both simple and complex programs. And most importantly, it can be used to relate the value-added of SIV&V between dissimilar programs.

3. Return On Investment (ROI)

In order to show ROI for SIV&V costs, the “I” in ROI must be quantified. SIV&V costs are dependent upon software program complexity, size, and needs, as well as, the breadth and depth of involvement required of the SIV&V team performing the service. As a general rule, total SIV&V costs are dependent upon the extent of SIV&V performed, and in which phase of the program SIV&V is begun.

An accepted management metric that is used to determine the worth of an investment is ROI. It is also known as the benefit-to-cost ratio. Even though there are those who believe that only a small impact is attained by SIV&V, there is sufficient evidence that numerous DoD and National Aeronautics and Space

Administration (NASA) software programs have implemented successful SIV&V programs that have delivered positive ROIs.

This study selected two programs, Systems B and D, from Table 3 “SIV&V Metrics” in Section A 2, in an effort to survey the ROI for the weapon systems employing SIV&V. Although many ROI calculations exist, this derived calculation would best fit our software situation.

The derived formula,

$$ROI = (A - (B - C)) / \text{Overall SIV\&V Costs}$$

Where: A = Costs avoided by intercepting defect

B = Costs to process the defect

C = Costs to correct the defect

is used to calculate the SIV&V ROI.

The general approach for calculating ROI was to first calculate the average cost per delivered source line of code (SLOC). Second, the development phase in which SIV&V detected the error was determined. For this study, all SIV&V detected errors found occurred in the “Test Phase.” It goes without saying, that waiting until testing to identify and remove problems negates most of the benefits associated with SIV&V. Third, a determination was made to find in which phase the error originated. Using multipliers based on documented, historical data, the cost to process and correct the defect was calculated. Two different multipliers were used in order to show that there is disagreement between the relative costs to correct defects by phase. Finally, the cost avoidance was determined assuming that the error would not have been caught until deployment if no SIV&V had been utilized. The inputs to the ROI calculation are addressed below.

a. Average Source Line of Code (SLOC) Development Cost

First, a basis for cost to repair a defect was needed. It was decided that the developer's cost per delivered SLOC would be used as an input for determining cost avoidance and defect correction. This relationship can be described as follows:

$$\$/SLOC = Total\ Development\ Software\ Budget / Delivered\ SLOC$$

Our study found that delivered SLOC costs vary based upon a variety of factors such as programming language, experience of the Developer, complexity of the system, and location of the Developer's organization. We found that the development cost for the systems analyzed was:

System B = \$275 per delivered SLOC

System D = \$125 per delivered SLOC

The important thing to note is that the delivered code costs can become much higher if SIV&V is not employed early in the lifecycle. An absolute dollar cost is not always the most important central aspect, but rather the costs of changes and errors to total cost which are phase dependent. Details of this research are demonstrated in the paragraphs below.

b. Analysis of SIV&V Software Trouble Reports Corrected and Implemented by the Developer

For the systems analyzed for this study, Software Trouble Reports (STRs) or Software Change Requests (SCRs) were accepted, analyzed, corrected, and implemented by the developer. The STRs were assessed as to how the developer classified each anomaly as to the phase in which the error originated. The classification of phase originated is shown in Tables 4 and 5 below:

Table 4. System B

Phase Originated	# STRs	SLOC
Requirements	9	24
Design	2	38
Code and Unit Test	48	567
TOTALS	59	629

Table 5. System D

Phase Originated	# STRs	SLOC
Requirements	74	4154
Design, Code, Unit Test	261	1440
Sys Test & Integration	12	49
TOTALS	347	5643

From the metrics in the above tables, one can identify the large difference in the number of STRs submitted by the different systems. System B was only 4.9% of the developer budget and System D was 8.0%. This data indicates that more problems are typically found when a larger SIV&V agent is established. Likewise, the data reveals where increased emphasis must be placed, namely Design, Code, and Unit Test phases. Based on our examples we believe that most programs would experience similar results.

c. Multipliers for Relative Cost to Correct Defects

Two different multiplier sources were utilized for this study. In Table 6 below, the first one shown was developed by Barry Boehm based on an old study of his which analyzed three different systems (Boehm 1989 206). The other source was from the Titan study (Barber 3).

Table 6. Study Multipliers

Phase Detected	Phase Originated	Boehm Multiplier	Titan Multiplier
Requirements	Requirements	1	1
Design	Requirements	3	5
Design	Design	2	1
Code & Unit Test	Requirements	9	10
Code & Unit Test	Design	6	2
Code & Unit Test	Code & Unit Test	1	1
SI Test	Requirements	29	50
SI Test	Design	26	10
SI Test	Code & Unit Test	20	5
SI Test	SI Test	1	1
Integration	Requirements	74	130
Integration	Design	71	26
Integration	Code & Unit Test	65	13
Integration	SI Test	45	3
Integration	Integration	1	1
Operation	Requirements	169	368
Operation	Design	166	64
Operation	Code & Unit Test	160	37
Operation	SI Test	140	7
Operation	Integration	95	3

The derived formula,

$$\text{SLOC DC} * \text{SLOC RC} * \text{PM} = \text{CPCD},$$

where SLOC = Source Lines of Code

DC = Defect Cost,

RC = Repair Cost

PM = Phase Multiplier

CPCD = Cost to Process and Repair Defects

is used to calculate the estimated costs to process and correct defects, and the avoidance costs of intercepting the defects.

Calculating the estimated costs to fix the STRs implemented by the Developer using the two different sets of multipliers yielded the following:

Estimated Cost to Process and Correct Defects

	<u>System B</u>	<u>System D</u>
<i>Boehm multipliers:</i>	\$3.5M	\$20M
<i>Titan multipliers:</i>	\$1.2M	\$28M

As stated previously, the assumption is made that had SIV&V not been employed, the defects included in this study would not have been detected until the system had been deployed, thus resulting in dramatically increased costs to correct. As was shown previously, two different sets of multipliers were used to calculate the cost avoidance figures:

Costs Avoided by Intercepting Defect

	<u>System B</u>	<u>System D</u>
<i>Boehm multipliers:</i>	\$28M	\$119M
<i>Titan multipliers:</i>	\$ 9M	\$203M

The deltas between the two examples are very large especially for System D. As you can see the benefits of having an SIV&V agent is worth the investment. As documented by Lewis, Figure 3 below, latent requirements and design errors can cost up to 36 times more to detect and fix during test and integration than if caught in the phase in which they were generated (Lewis 1992 279). In retrospect, if SIV&V resources were spent on all of the development phases, dramatic reduction in costs could have resulted and without question, the worth of SIV&V would be justified as a cost saving mechanism and declared “cheap issuance” for the development program.

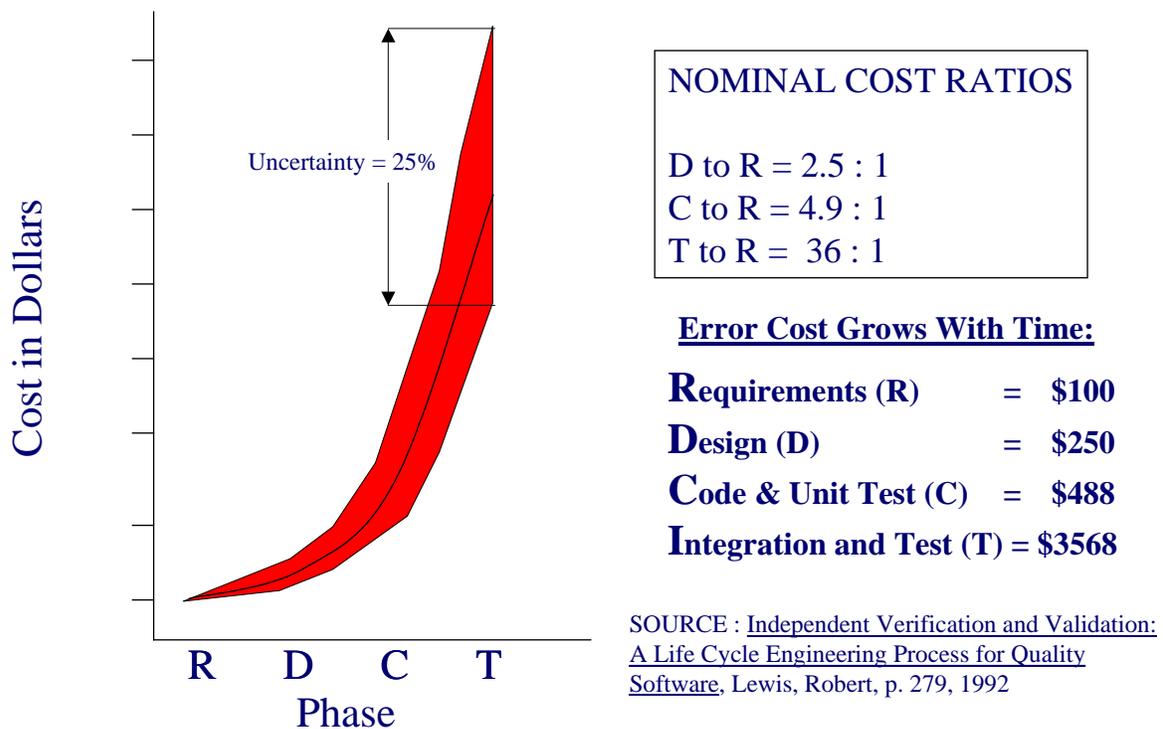


Figure 3. Average Cost of Discovering Errors (Lewis 1992 279)

d. Return On Investment Calculation

The growing interest to measure Return On Investment (ROI) is fueled by the need to document and measure improvements in both individual and agent performance. The resulting ROI calculation of systems B and D is as follows:

Return On Investment

	<u>System B</u>	<u>System D</u>
<i>Boehm multipliers:</i>	6.8	4.9
<i>Titan multipliers:</i>	2.1	8.8

Based on the ROI quoted in the Titan study of an ROI of 10, it is not unreasonable at all for the U.S. Army's Software Engineering Directorate (SED) to state for the systems analyzed that the ROI range is:

$$2.1 < SIV\&V\ ROI < 8.8, \text{ with an average of } 5.5:1$$

If the sampling size was increased for all of the SED SIV&V programs, a more representative assessment could be made to the overall ROI that is delivered. This report does conclude, however, that the ROI average for SED SIV&V efforts is likely within the recommended range. The software industry has yet to define a recommended operating range for ROI; however, a 2% ROI is the projected break-even point. An ROI greater than or equal to 5% significantly justifies the effort and results in cost improvements, increased quality, and enhanced schedule.

Another way to look at ROI is the differences between fixed and variable costs. Typically, costs for SIV&V agents are fixed because the SIV&V agent is focused on prevention. The developers cost are mainly variable because the developer is focused on detection and correction. If the SIV&V effort increases and prevents increasing numbers of defects then SIV&V has helped to reduce variable costs. As a result, investments in SIV&V should continue as long as the agents fixed costs remains below the prevented defects cost (B +C, variables defined in ROI section above).

e. *Development Schedule Reduction*

The Standish Group examined 8,380 software projects and found that 35% of all software efforts were “challenged,” 16% were successful, and 31% cancelled. The “challenged” groups exceeded software delivery schedules by 222%, were over budget by 189%, and were missing approximately 39% of the expected capabilities (NASA 1985 2). Given this type of track record, the question that many Program Managers (PMs) ask is what can they do to improve the software product and reduce the development schedule? The answer is SIV&V.

Software SIV&V is indeed an important aspect of developing quality software (Defense Acquisition University 387). History has shown that software defects have delayed multi-million dollar space launches, prevented the Denver airport opening for years, destroyed NASA Missions (Mars Climate Orbiter and Polar Lander), killed service men and women (e.g. marines in a helicopter crash), and shutdown banking and emergency response systems.

A majority of the time, SIV&V does not begin until the testing phase of an acquisition program, where problems become more noticeable. Waiting until the end to identify and report problems only increases the development schedule and cost. A simple analogy from the auto industry can be used here to illustrate this issue within the software world. “The mass-producer...”, in this case the software developer, “...keeps the [assembly] line moving at all costs but ends up doing massive amounts of rework at the end, while the lean producer...”, in this case a PMO using SIV&V, “...spends more effort up front correcting problems before they multiply and ends up with much less total effort and higher quality in the end” (Womack 116).

A study published in 2002 by the National Institute of Standards and Technology (NIST) estimated that software bugs are so common that their cost to the American economy alone is \$60 billion a year or about 0.6% of gross domestic product. According to NIST, 80% of the software development costs of a typical project are spent on identifying and fixing defects (Building A Better

BugTrap 2). A bug which costs \$1 to fix on the programmer's desktop costs \$100 to fix once it is incorporated into a complete program, and many thousands of dollars if it is identified only after the software has been deployed in the field (Building A Better BugTrap 2).

SIV&V's main goal is to complement the development effort by reducing risks and identifying errors, which often lead to schedule slippage. PMs will receive the greatest payoffs when they utilize a SIV&V agent for thorough requirements and design verification aimed at preventing otherwise costly errors, omissions, and inadequacies from ever reaching the coding stages.

Further, SIV&V can be very instrumental in finding errors during the coding phase. On average, professional coders make 100 to 150 errors in every thousand lines of code they write, according to a multiyear study of 13,000 programs by Humphrey of Carnegie Mellon (Mann 3).

Implementing SIV&V throughout the life cycle will lead to better software products, potentially reducing schedules, and costs, in addition to saving lives. If applied early and effectively, it can help maintain balance in the cost-schedule-quality equation throughout the development process (Callahan). If applied late or reluctantly (so-called "11th hour V&V"), it can fail to have any effect and be cost-ineffective (Callahan). Embracing SIV&V is an endeavor to reduce the risks and associated cost-schedule pressures inherent in the development of complex software, by changing the current cultural paradigm of "mass production" to one of "lean production."

f. More Measurable Results of SIV&V

In addition to calculating a ROI for SIV&V programs at SED, valuable data can be gleaned from a close look at a large SIV&V effort at SED with 20+ years of historical SIV&V data available. This program, referred to as System D for this report, has had much success with SIV&V, and consequently, has had much success as a program. System D's SIV&V effort is about 8-10% of the Developer's yearly budget, including labor, travel, and material. System D's SIV&V is in-phase, has source-level access, reports directly to the

Government customer, and has an enhanced SIV&V scope. System D's SIV&V effort has been ongoing since the early requirements development stage of the system. These factors, combined with the benefits of the SIV&V being performed by SED, which is a Government Life Cycle Center, have contributed to the success of the SIV&V effort as well as the program itself, as demonstrated by the following information.

System D's SIV&V team found and reported errors on 44% of requirements SLOC that were deemed completed by the Developer. Since the SIV&V is "in-phase," these requirements errors were able to be corrected during the requirements phase. A quick look at the "Study Multipliers" in Table 6 shows the rapid growth in cost to correct that 44% of new requirements SLOC as the error is propagated through the development cycle. At best, if all of the errors were found and fixed in the next phase, the cost would have increased by a factor of five. Near the worst case, if all errors were propagated through integration, but discovered before operational fielding, the cost would have increased by a factor of 130. If carried into operational fielding, the factor grows to 368. One must also consider how many of the requirements errors might have gone undetected altogether and what the resulting cost would be in dollars, schedule, and user benefit. However, due to the iterative nature of the developer's build cycle, the requirements phase is not closed until delivery of requirements to, and return from, the SIV&V team; therefore, the maximum number of requirements errors can be found and fixed during the requirements phase. Approximately 44% of the developer's new requirements SLOC would have had errors that would have been discovered and fixed at much higher cost and schedule impact.

Similarly, System D's SIV&V team found and reported errors on 1.2% of SLOC during the Design, Code, and Test phase that were deemed completed by the Developer. Again looking at the "Study Multipliers" in Table 6 it is easy to quickly spot the value of finding and correcting these errors in the phase in which they were generated, or quickly thereafter. Also, note that this

percentage of SLOC with error is artificially low due to reporting methods. Further down the development cycle, System D's SIV&V team found and reported errors on 0.8% of SLOC during the System Test and Integration Phase. Once more, this percentage is artificially low due to reporting and tracking methods of the developer, but provides insight into the effectiveness of SIV&V.

In 2001, NASA conducted a study on "Developing Risk-Based Financial Analysis Tools and Techniques to Aid IV&V Decision-Making." The study indicated that due to SIV&V's presence the software product improved tremendously. By inserting SIV&V into the requirements phase, the software developer had to rework 22% of its effort due to defects. Both, the design and programming phases required 7% rework. However, what was most telling and shocking was when the SIV&V effort was delayed until the test and integration phase, 28% rework had to occur by the developing organization to correct the product.

There are many other benefits the System D SIV&V team has provided to the Government customer, which cannot be easily tracked to a metric, but give valuable payback. System D's SIV&V team is often used as a source of expertise by the Government customer. Since the System D SIV&V team was put in place very early in the system development, and has maintained an unusually low personnel turn-over rate, both the system-level and functional-level of expertise of the SIV&V team meets or exceeds the level of expertise of the developer. Having a second source of expertise, in addition to the developer, is valuable in three ways, 1) when the developer loses resources, 2) when it is more efficient to have the SIV&V team work an issue to prevent schedule impacts to the developer, and 3) when the SIV&V team is a cheaper resource than the developer.

System D's SIV&V team has been relied upon heavily during times of war and when the soldier in the field needs an explanation, since the software developer is typically no longer available, and for special studies and investigations. Since SED, a Government institution is System D's SIV&V

resource, the Army Evaluation Command relies on the SIV&V team for testing requirements that are outside of their resource limitations. System D's SIV&V team also represents the Government customer on the System Safety Analysis Board.

So when taking a close look at an example of SIV&V in action, the value of SIV&V, when implemented properly, is measurable in both cost and intrinsic value, and both values are large. System D's example of success also demonstrates several crucial keys to successful SIV&V. Historical data from many sources demonstrates that SIV&V does, in general, have a positive ROI. Combining "classical" SIV&V with lessons learned and crucial keys from the rich history at SED can drive the ROI to its maximum value.

g. Examples With and Without SIV&V

(1) Ariane 5 Without SIV&V. On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff. Media reports indicated that the cost of this loss was half a billion dollars -- uninsured (Knutson; Missile Defense Agency 24).

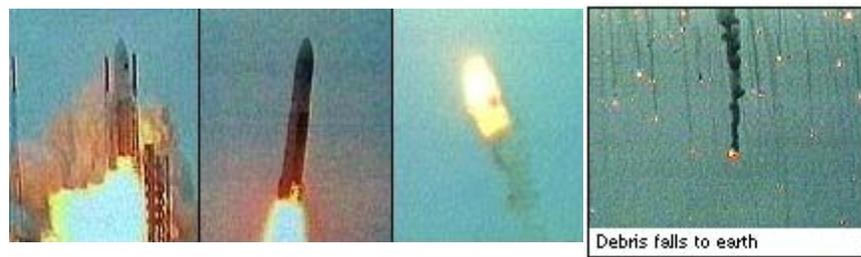


Figure 4. Ariane 5 Crash (Knutson; Missile Defense Agency 24)

Ariane-5 was the newest in a family of rockets designed to carry satellites into orbit. On its maiden launch on June 4, 1996, it flew for just under 40 seconds before self-destructing, destroying the rocket and its payload of four satellites (Knutson; Missile Defense Agency 24).

After the incident, Ariane immediately set up a Board of Inquiry to conduct a thorough investigation to discover the root cause of the accident (Knutson; Missile Defense Agency 26).

The core problem in the Ariane failure was incorrect software reuse. A critical piece of software was reused from the Ariane-4 system, but behaved differently in the Ariane-5 because of differences in the operational parameters of the two rockets. One of the important lessons from the Ariane-5 failure is that the quality of a device's software must be considered in the context of the entire system. It is an important lesson to keep in mind as software reuse continues to be an important trend in software engineering. If SIV&V were utilized these abnormal conditions could have been avoided (Knutson; Missile Defense Agency 26).

The Ariane failure was highly publicized and documented.

(2) NASA With SIV&V.



**NASA
POLICY
DIRECTIVE**

Directive: NPD 8730.4
Effective Date: August 01, 2001
Expiration Date: August 01, 2006

This Document Is Uncontrolled When Printed.
Check the NASA Online Directives Information System (NODIS) Library
to verify that this is the correct version before use:
<http://nodis.hq.nasa.gov/Library/Directives/NASA-WIDE/contents.html>

Responsible Office: Q / Office of Safety and Mission Assurance

Subject: Software Independent Verification and Validation (IV&V) Policy

1. POLICY

NASA will conduct Independent Verification and Validation (IV&V) based on the cost, size, complexity, life span, risk, and consequences of failure. Specifically, NASA will:

- a. Establish and apply criteria, tools, and methodology to evaluate and assess software risk for the purpose of identifying the appropriate level of IV&V.
- b. For programs and projects governed by NPG 7120.5A, task the NASA IV&V Facility in Fairmont, WV, to manage the performance of all IV&V for software identified per the established criteria, and for any other safety critical software (as defined in NASA-STD-8719.13A).
- c. Require programs and projects governed by NPG 7120.5A to determine the level of IV&V to be performed with the explicit involvement of the NASA IV&V Facility.
- d. Require NASA programs and projects that contain mission or safety critical software to document decisions concerning the use of IV&V.

Cost of Failure is the Rationale for NASA Support to SW IV&V

	'93	'96	'97	'98	'99
Airbus A320					
Ariane 5 Galileo Poseidon Flight 965					
Lewis Pathfinder USAF STEP					
Zenit 2 Delta 3 NEAR					
DS-1 Orion 3 Galileo Titan 4B					
Aggregate Cost:		\$640 million	\$116.8 million	\$255 million	\$1.6 billion
Loss of Life:	3	160			
Loss of Data:		◆	◆	◆	◆

Figure 5. NASA SIV&V Policy (Missile Defense Agency 23)

h. DoD SIV&V Examples:

(1) AEGIS Without/With SIV&V. Recent headlines stated “AEGIS SHIPS USS VICKSBURG and USS HUE CITY Docked After Contractors Have ‘Free-Reign’ of Software - Deemed Not Seaworthy Until SIV&V is Reinstated” (Missile Defense Agency 27), and “Software Glitches Leave Navy Smart Ship Dead In The Water” (Slabodkin).

The cost for NSWCDD's Aegis Baseline SIV&V process varies significantly based on the size and complexity of the functional capability that is being developed. A software development effort such as Baseline 6 Phase 3 costs as much as \$50M LOE spread over the design, code, integration and test phases in a 4-5 year span for the SPY-1 (Radar) element alone. Numerous significant issues and problems were identified, investigated, and resolved prior to major milestones as a result of NSWCDD's SIV&V involvement. Considering that SPY-1 Baseline 6 Phase 3 cost well over \$500M to develop and field, \$50M for SIV&V was a small price to pay (Missile Defense Agency 27).

The yearly cost NSWCDD applies to SPY-1 SIV&V for SW development is about \$10M. This includes personnel to support design reviews, code inspections, unit testing, test procedure development, formal element testing, formal system level testing, SW configuration management, SW quality assurance, SW documentation, training, and facility operations (Missile Defense Agency 27).

(2) THAAD Without/With SIV&V. After Three Repeated Flight Test Failures (Dem/Val), THAAD Project Office requested SIV&V to test all software changes to verify correct software implementation.



Figure 6. THAAD Electronics Box (U.S. Army 28)

The THAAD missile SIV&V effort identified 10 percent (approximately 200) of all avionics software problems.

- Many critical; fixes had to occur before flight tests resumed
- SIV&V found these problems in released (tested) code
- SIV&V testing helped get the THAAD program on track

i. Commercial Examples: No SIV&V Agent Utilized

(1) FAA. The Federal Aviation Administration (FAA) encountered a software glitch; software patches were dispatched to Boston and other airports to enhance the ground-based radar systems. The new software failure was noted when the system could not see two planes approaching each other on the runway, which was a failure caused by the patch. The FAA reported in another incident that a backup system that was designed to handle planned server downtime resulted in a three hour loss of air traffic control communications between 800 plus airplanes (Entries Hall of Shame).

(2) AT&T. American Telephone and Telegraph (AT&T) experienced a billion dollar failure in January 1990, when a software failure took down the entire United States (US) long distance telephone network for nine hours (AT&T Wireless).

(3) US Airways. US Airways experienced a software failure in April 2005, when the ticketing system issued incorrect fares for several hours. During this occurrence some tickets were sold for under \$2.00. The airline honored the reduced fares as a gesture of good faith, but lost a substantial amount due to the failure (Entries Hall of Shame).

(4) Toyota. In October 2005, the Toyota Motor Company recalled more than 75,000 Toyota Prius-hybrids due to a software failure that may have shut down the engine. Toyota quickly implemented the recall which avoided having the defect become permanently associated with the vehicle line or with hybrid safety (Entries Hall of Shame).

C. APPLYING SIV&V IN THE CYCLE PHASE

The following sections are primarily based on an IV&V process chart authored by R. O. Lewis (Lewis 1998).

1. Introduction

Software Independent Verification and Validation (SIV&V) activities (notional list of activities shown in Figure 7 below) are used to assess risk and the quality of software throughout the development process and are performed at the unit, module, integration, and system levels. SIV&V activities should begin early in the software development process to assure consistency between product specifications and requirements, design, implementation and testing.

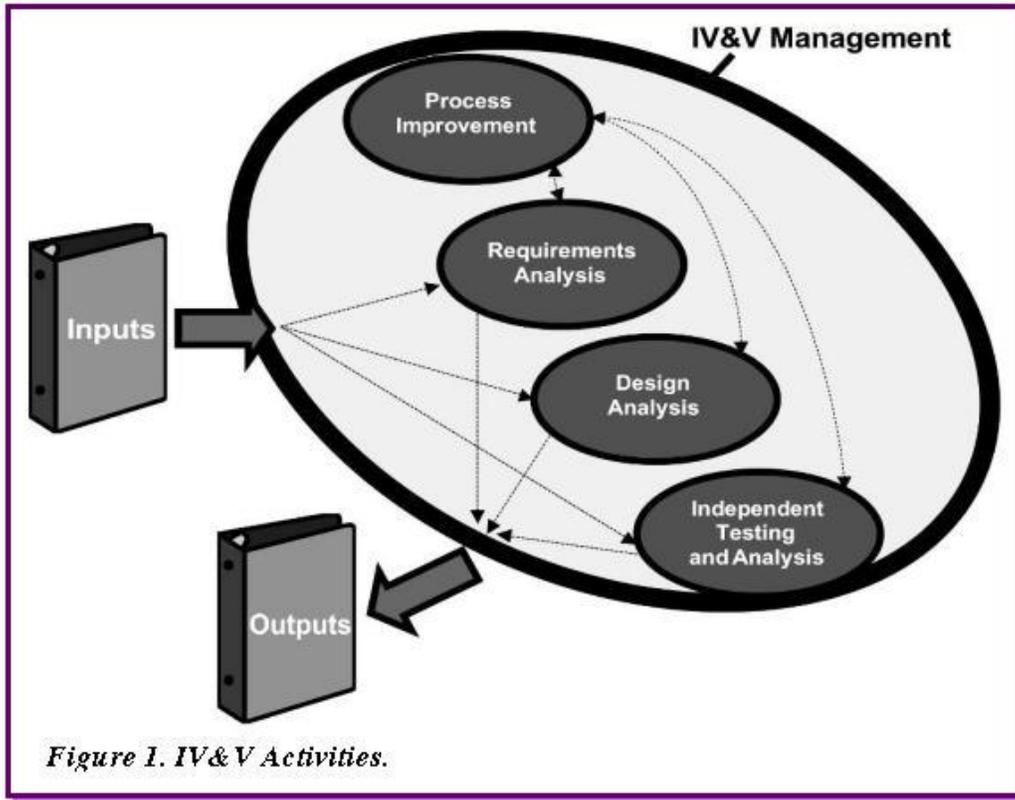


Figure 7. Notional SIV&V Activities (Walters)

Independent reviews are conducted during and at the end of each phase of the life cycle to determine whether established requirements, design concepts, and specifications have been met. The customer relies on the Independent agent to provide a go or no-go decision on whether to proceed to the next step or start the process over. The following sections will highlight some of the different activities that are required of SIV&V in each cycle phase.

2. Concept Definition Phase

Concept Development is typically the first phase of a major system development program. Normally, this phase includes competition among contractors who offer different solutions to some basic need, in which case they often either produce prototypes of their designs or conduct studies and analyses. Although, the process may vary significantly, programs that have a formal SIV&V effort at this point in the life cycle, invariably end up improving the System

Specification (SS), analyze the external interface requirements and associated inputs, and examine the feasibility and adequacy of each competing design. SIV&V reviews all available input materials to enhance its program understanding as indicated by the following tasks (Lewis 1998):

- Initial SIV&V activities focus on evaluation of the mission needs, external system requirements, external interfaces, and overall program planning data. This should include a survey of the users requirements, examination of make-buy decisions, assessment of technology drivers, and evaluation of the operational concept.
- The various program plans are assessed for consistency, completeness, and correctness to ensure essential aspects of the conceptual system are addressed. These include but are not limited to: interface control, configuration management, safety, risk management, master program plan schedules, integration, resource control, environment, and planning for reaching operational capability.
- SIV&V evaluates the system concepts against the requirements mission and user needs. This information is used to assist in the evaluation of the Technical Requirements Document (TRD), system requirements, and drafts of the SS or Prime Development Specification (PIDS) produced and controlled by the customer.
- A System Requirements Review (SRR) covers each draft release of the System Specification. There are often several SRRs held to firm up the requirements.
- The SIV&V Plan is drafted as early as possible, but usually has to await the generation of the contractor's Software Development Plan (SDP) for completion; because, much essential information is still missing.
- As the program requirements become stable, SIV&V examines program feasibility and completeness, allocation of requirements to major subsystem elements, and supports the initial conversion of the

critical functions and requirements into a formal listing or requirements trace database. Other products, such as the discrepancy logs are developed and captured during this period.

- As the SIV&V team identifies potential problems and issues, reports are submitted to the customer's Configuration Management (CM) point or designated customer representative. These problems or anomalies are typically reported and summarized in status reports.
- Engineering analysis is used to evaluate the evolving systems concepts. If the system is extremely complex, high level modeling and simulation are often used to evaluate the completeness and feasibility of competing system designs.
- SIV&V identifies inconsistencies and shortcomings in concepts, technology drivers, make-buy decisions, documentation, trades studies, and the evolving System Specification. Special emphasis is placed on critical software issues including: user and performance requirements, operational feasibility, modeling and prototyping, hardware performance needs, safety issues, and system and software risks and their mitigation.

SIV&V Battle Rhythm (Concept Development Phase)

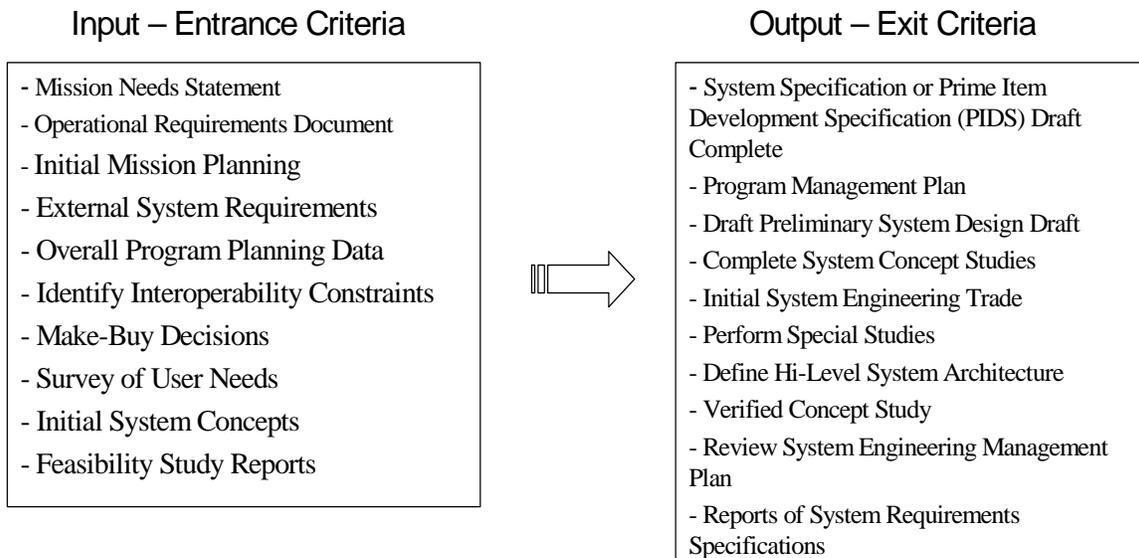


Figure 8. SIV&V Battle Rhythm Concept Development Phase (Lewis 1998)

3. Requirements Phase

“During the requirements phase, the user tries to articulate a concept of expected system function and performance into concrete detail” (Department of Air Force 2-20). SIV&V activities during the requirements phase include analysis of the software and hardware requirements to determine if the software engineers have translated user definitions into realistic and achievable specifications. Similarly, SIV&V determines if the established requirements are testable and capable of being satisfied. The requirements phase is a segment that first concentrates on system requirements, and then software requirements. The activity verifies that the software requirements have been prepared in accordance with applicable standards, and evaluates the progress of the requirements toward achieving an operational system. Requirements Analysis identifies critical risks and potential process and product improvements. Potential risks due to defects in the requirements are identified and addressed through discussions with the developer. High priority risks are analyzed to

identify potential process and product improvements. This multi-part requirements phase is in concert with most approved development standards and typically encompasses the following tasks (Lewis 1998):

- During concurrent engineering, hardware and software requirements are synthesized to include numerous trade-offs and performance considerations. Hardware benchmarks are used to measure performance using typical applications and are verified by SIV&V.
- This phase addresses the following basic verification criteria, namely system and software requirements verification, and evaluation of the initial test plan and the SIV&V test plan. This phased approach accomplishes additional strategic tasks more or less as follows:
 - Inputs from Concept Development Phase plus those listed above.
 - Initial steps involve verifying the operational, functional, performance, and program requirements, and assessing and racking the critical functions including the use of SIV&V requirements tracking database or tool.
 - Plans and planning factors are evaluated to ensure that SIV&V works in lock step with the development team.
 - Evaluate the engineering trades to ensure feasibility and completeness of the requirements documents, which can be measured quantitatively.
 - Examine anticipated behaviors and performance needs of the system and its software to develop initial Measures of Performance (MOPs).
- The System Specification is verified and analyzed for completeness, consistency, testability, correctness, understandability, and feasibility. If a System Segment Design Document (SSDD) is produced, it is verified as well.
- Prototypes or models of the system architecture are evaluated.
- SIV&V participates in the System Design Review (SDR) and Critical Design Reviews (CDR).
- SIV&V assesses the development standards and guidelines used. This includes review of the draft and final Software Development Plan (SDP).

- During this time, program data collection begins in earnest and feeds the SIV&V status report. Early metrics typically include requirements stability and adequacy. Risk tracking begins and the highest risks items are routinely monitored. SIV&V program status and performance metrics augment the status reporting.
- Following the SDR and the functional base-lining of the System Specification, SIV&V emphasis shifts to software requirements and to the extent necessary, hardware requirements.
- As the interface definitions mature, SIV&V reviews and evaluates the Interface Requirements Specification (IRS) and Interface Control Document (ICD). Agreement between these documents and the SSDD is vital.
- The allocation of functions and requirements to Hardware Configuration Items (HWCI) and Software Items (SI) is verified to provide the foundation for in-depth verification of each SRS for each SI.
- SIV&V participates in the Software Specification Review (SSR) to ensure that each CSCI is ready to transition into the architectural design.
- It is here that analytical methods, tools, and techniques are used effectively to evaluate and verify the requirements.
- Hardware, support software, and tools selected for the Software Engineering Environment (SEE) are evaluated and recommendations are provided.
- Special emphasis is placed on critical software issues.

SIV&V Battle Rhythm (Requirements Phase)

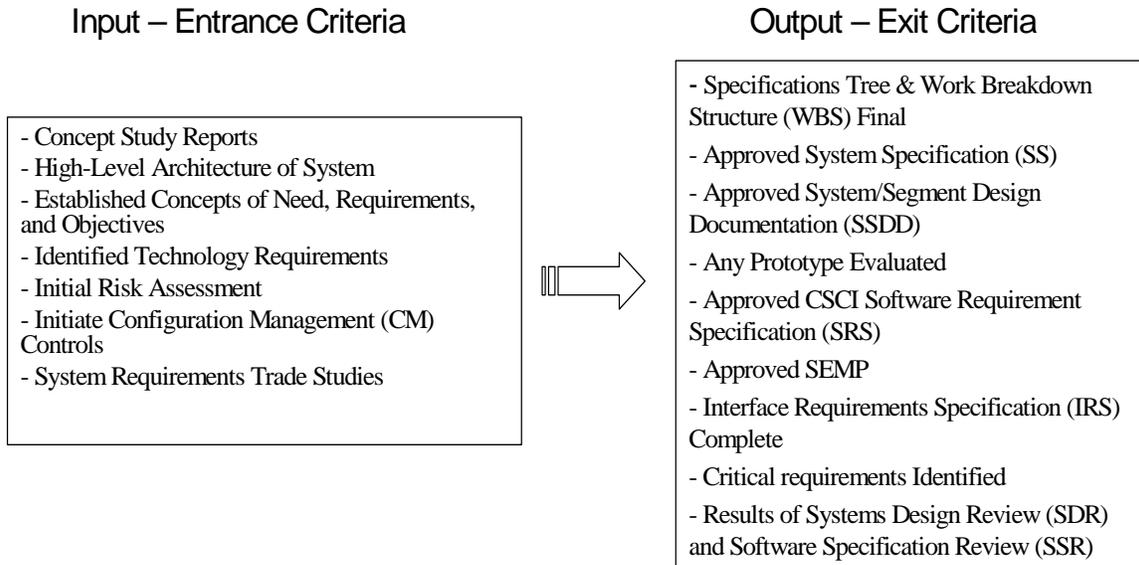


Figure 9. SIV&V Battle Rhythm Requirements Phase (Lewis 1998)

4. Design Phase

The Design Phase activity is conducted by reviewing the software design products (e.g., object models, sequence diagrams, algorithm descriptions, design specifications, etc.) of each software baseline build and evaluating the products for adherence to software requirements. The design products (e.g. Software Design Documents (SDDs)] are also analyzed for consistency, completeness, and correctness with respect to the system, software, and interface specifications. Software problems identified in the Design Analysis activity are documented as risks, and comments are provided to the developer for adjudication. Typically, the Design Phase is multifaceted, meaning that it consists of a Preliminary Design Phase or Architectural Design Phase and a Detailed Design Phase. These facets and SIV&Vs input/output processes are discussed below (Lewis 1998).

a. *Architecture Design*

Architectural design verification gives attention to the soundness and completeness of the software design, interface design, and traceability of the requirements into the design. The following tasks are typical of this phase (Lewis 1998):

- This process consists of a number of steps that evaluate the allocation of functions and capabilities especially centered on critical functions, a thorough examination of the architectural design and its documentation in the preliminary Software Design Document (SDD), and verification of the system architecture.
- The SIV&V team evaluates the fidelity of the design to ensure adequacy of algorithms given the performance expected from the elected platform. This is accompanied by analysis of the operating timeline estimates.
- The architectural design is evaluated for its completeness, modularity, efficiency, complexity, stability, and understandability.
- The interface requirements are tracked from the Interface Requirements Specifications (IRS) into the SIs and HWCIs and into the Interface Control Document (ICD). The interfaces are evaluated for completeness, consistency, correctness, and ability to implement.
- Externally supplied data is assessed for adequacy and completeness, and is certified, when necessary.
- Internal data is analyzed using data flows generated by selected software tools. A data dictionary is produced by the developer, which is evaluated by SIV&V for accuracy and completeness.
- SIV&V proven methods, tools, and techniques are selected and used for analysis and evaluation.

- Test requirements are extracted and traced to the developer's Software and Hardware Test Plans (STP and HTP). This ensures adequacy of the test plans and also provides data for development of the SIV&V Test Plans when independent testing is required. Many systems today are networked and require extensive 'distributed' test environments; it is increasingly difficult and cost prohibitive to attempt to exactly replicate these facilities for SIV&V. In such cases, 'dual-use' testing is encouraged in which SIV&V and developers share facilities and conduct joint testing.
- User requirements are verified, and safety factors and the Failure Mode Effects Analysis (FMEA) are evaluated.
- Change requests to the design and requirements are tracked, coordinated, and evaluated for impact to the design documentation.
- The architectural design portion of the SDD is thoroughly evaluated for completeness, consistency, traceability, correctness, implement-ability, and testability.
- SIV&V participates in the Architectural or Preliminary Design Review (ADR or PDR).
- Human engineering factors, command and control features, and functions are evaluated to ensure adequate user and operator interaction.
- All critical functions are traced, verified, and documented in the SIV&V requirements tracing database.
- The issue Tracking Log tracks the status of all essential open issues.
- The SIV&V Test Plan is drafted and may become an extension of the developer's Software Test Plan (STP), if dual-use (joint) testing is conducted.

- Development changes that occur during this phase are immediately fed into the appropriate verification activity and assessed for their impact.

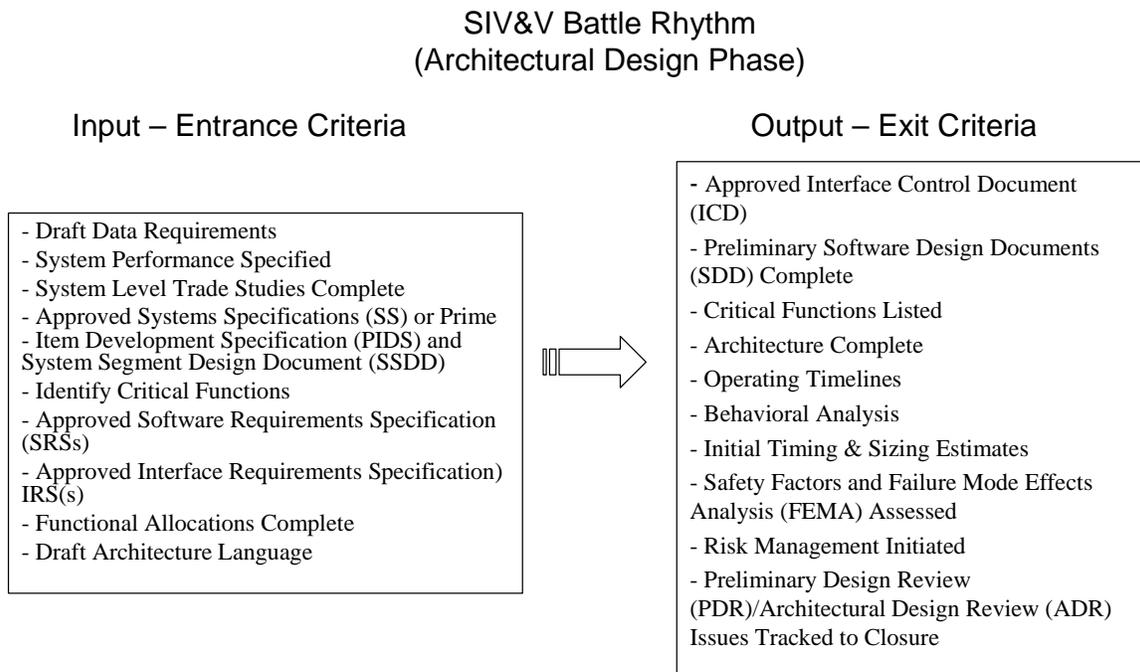


Figure 10. SIV&V Battle Rhythm Architectural Design Phase (Lewis 1998)

b. Detailed Design

Detailed design verification examines the detailed design of the software for completeness, consistency, logical and functional correctness, implement-ability, and feasibility and typically addresses the following tasks (Lewis 1998):

- SIV&V examines and assesses the design for efficiency, modularity, fidelity, complexity, testability, clarity, flexibility, and stability.
- Metrics and performance indicators are collected, assessed, and reported back to the customer on program maturity and potential delinquencies.

- SIV&V participates in design inspections hosted by the developer and customer to evaluate the thoroughness and discipline of the design team; this is especially true when integrated product teams are used.
- In view of the fact that detailed design of complex systems is invariably supported by software design tools, SIV&V evaluates them and identifies, selects, and uses portions of the developer's tool set to ensure repeatability; additionally, specially chosen complementary SIV&V tools are used to enhance analysis and error detection; and further investigate suspect designs.
- SIV&V performs several verification activities in parallel, that include hardware/software mapping; verification of key algorithms; analysis of control flow, schema, and behaviors; and detailed timing and sizing analysis. Algorithms are selected based on criticality, complexity, and performance, and are thoroughly evaluated, often by coding and executing them on a tool-bearing host to determine their accuracy, performance, and suitability.
- The Interface Design Document (IDD) and Interface Control Document (ICD) are evaluated for consistency, completeness, and accuracy. This includes hardware interface assessment to ensure integrity of the design.
- SIV&V verifies the developer's Software Test Descriptions (STDs) and Software Test Plans (STPs). This process ensures adequacy of both the development and SIV&V test programs to fully verify all critical requirements and functions. When dual-use (joint) testing is employed, SIV&V feeds its discrete test requirements, test cases, and data collection needs to the developer for inclusion in its testing. This technique is used

when it is impractical, too expensive, or impossible to precisely replicate the development configuration.

- SIV&V thoroughly evaluates the user and operator interfaces and command and control interactions, and often participates in safety assessments of the software.
- Execution time budgets and task schedules are assessed to estimate the adequacy of operating margins.
- SIV&V verifies the completed SDDs based on virtually all detailed design activities discussed in this section.
- Continue verifying and tracking critical functions using the SIV&V requirements tracking database and critical function list.
- SIV&V evaluates the visualization and the Graphical User Interfaces (GUIs) proposed for use or prototyped for the system—screens, human machine interfaces, controls, use of color, mimics, and so forth.
- SIV&V participates in hardware evaluation and design to benefit the software design verification. A strong understanding of the hardware is essential to software SIV&V.

SIV&V Battle Rhythm
(Detailed Design Phase)

Input – Entrance Criteria

- Architectural Design Complete per Software Design Document (SDD)s and Tool Output
- Preliminary Timing and Sizing Data
- Design Trade Studies
- Commercial Off-The-Shelf (COTS)/GOTS Selection
- Input/Output (I/O) Data Sources/Definitions
- Development Environment Software Engineering Environment (SEE) Established
- Draft Interface Design
- External Design Constraints
- Functional and Performance Requirements Database (DB)



Output – Exit Criteria

- Detailed Designs Complete per Critical Design Review (CDR)/Detailed Design Review (DDR)
- Agreement on Critical Functions
- Complete Software Design Documents (SDDs) for each Computer Software Configuration Item (CSCI)/Software Item (SI)
- Complete Interface Design Document (IDD)
- Complete Database Design Document (DBDD)
- Verified Operating Timelines
- Risk Management Performed
- Closure on All Critical Design Review (CDR)/(Detailed Design Review (DDR) Issues
- Approval for Coding

Figure 11. SIV&V Battle Rhythm Detailed Design Phase (Lewis 1998)

5. Code and Development Testing Phase

Code verification is commonly a three-phase activity used with most development practices. This forms a suitable way of viewing the evolution of code from the Computer Software Unit (CSU) (smallest compleiable entities), to the Computer Software Component (CSC) (collections of units that have common or supporting functions), to Software Item (SI) level and supports the testing. The Institute of Electrical and Electronics Engineers (IEEE) J-STD-016 and others handle these steps as two parts, in which “Software Units (SUs)” comprise everything below the SI level. Despite the number of discrete levels of coding the SW goes through, this pattern depicts the growth from small chunks of code to components (tasks and packages) and finally to complete Sis. The following tasks are typical of this phase (Lewis 1998):

- SIV&V typically begins code verification at the level where execution of the code can occur. It is often too hard to force a CSU to execute and do something representative, so this level of verification is usually avoided as it is too costly and time consuming. The converse is true at the CSC, package, and SI level. Beginning with CSC, the code can usually perform one or more complete tasks, and therefore, can be evaluated and verified in a meaningful way. There are many variations to this approach resulting from differences in software languages, developer preferences, maturity of the software, stability of the requirements, and the like.
- One of SIV&V’s favored approaches is to use a code analyzer (i.e. McCabe, FlexeLint, and Vector Cast) on CSCs and SIs. These tools find a large number of the embedded coding errors, and dead code, inspect the code for standards and rules violations, uncover bad coding practices defined by industry (i.e. Carnegie Mellon Software Engineering Institute), and semantic errors. These tools are able to very quickly calculate the complexity of code, and help point the SIV&V analyst to the hot spots to make assessments and look for possible

problems. These tools can support static and dynamic execution of software on most platforms. A real plus is that these tools are very fast and yield consistent results after repeated runs, something human analysts cannot always do.

- In addition to the code integrity checks described, SIV&V completes the verification of the SDDs, IDD, ICD, and the design tools' outputs to ensure that the designs, databases, and interface documents match the code.
- Often times, SIV&V participates in code inspection, peer reviews, and walk-throughs.
- SIV&V ensures that the standards and coding practices described in the developer's Software Development Plan (SDP) are being followed.
- SIV&V evaluates the adequacy of the developer's test procedures and the test facilities and environment. This includes simulations, drivers, simulators, and database analysis tools.
- SIV&V then generates test procedures and conducts independent testing (i.e. "white box/black box" testing) to complement testing being performed by the developer, or as previously described, may input its requirements into the developer's STD for dual use testing.
- SIV&V continues tracking and focusing on the critical functions to ensure the highest possible degree of safety, reliability, and user and mission responsiveness.
- This phase provides an opportunity to verify the networking for the test environment to support SIV&V. This is especially important when the developer and SIV&V teams are separated geographically.
- Software change activity is typically heaviest during this phase and the next phase, stressing the Configuration Management (CM), problem tracking, and reporting systems.

**SIV&V Battle Rhythm
(Code & Development Testing
Phase)**

Input – Entrance Criteria

- Critical Design Review (CDR)/Detailed Design Review (DDR)
- Software Design Documents (SDDs) Complete
- Database Design Document (DBDD) Complete
- Interface Design Review (IDD) Complete
- Interface Control Document (ICD) Complete
- Detailed Design Compared to Code
- Software Testing Environment Available
- Developer Tests & Results Shared



Output – Exit Criteria

- Computer Software Unit (CSU), Computer Software Component (CSC), Computer Software Configuration Item (CSCI) or Unit/Item Requirements Verified
- Updated Allocated Baseline—Code and Documentation
- All Critical Requirements Traced to Code Level
- Approved Test Procedures
- Computer Software Unit (CSU), SCS & SCS or Unit/Item Test Results
- Test Readiness Review (TRR) for each Computer Software Configuration Item (CSCI)/Item
- Software Statistics

Figure 12. SIV&V Battle Rhythm Code and Development Testing Phase (Lewis 1998)

6. Hardware and Software Integration Phase

Hardware and Software Integration (HSI) verification testing has taken a major shift away from the pure development environment to the integration environment. Here the software is typically developed on Functional Equivalent Units (FEUs) and then moved to the actual hardware laboratory for integration, and finally installed on the Field Unit Equipped (FUE) hardware for eventual use. HSI usually means that a different organization assumes the responsibility for the software products. It also means that the software will run on actual mission hardware or on precise hardware equivalents for the first time. The following tasks are typical of this phase (Lewis 1998):

- SIV&V begins this phase by reviewing the integration and test planning documents. These typically describe the hardware environment in

which the software is to be integrated. Second, SIV&V reviews the developer's test documentation and reports to determine where to best focus SIV&V resources.

- SIV&V monitors the integration process and testing, and given the opportunity, performs or shares independent integration level testing to confirm the results reported by the integrator, and explore new facets and behaviors of the software, with special emphases on critical functions to ensure their adequacy, correctness, and integrity.
- In addition to performing SIV&V testing, an assessment is made of the adequacy and completeness of developer and SIV&V test facilities and support environments.
- SIV&V selects and uses methods, tools, and techniques for analysis and evaluation.
- An extremely important activity is the verification and analysis of the Software Product Specification (one per SI) including the source code. This is usually performed in preparation for the Functional Configuration Audit (FCA).
- The Functional Configuration Audit (FCA) is usually held when hardware and software integration is complete and the SI satisfies all of the requirements levied upon them in the Software Requirement Specifications (SRSs). However, this audit can, in some cases, be postponed until the next phase, following the Formal Qualification Tests (FQTs) and/or occasionally following a series of flight, and live tests, if required by the contract.
- In any case, product base-lining cannot take place until these audits have been held and all essential open items are closed, otherwise deferred, or waived.
- SIV&V also confirms timing, sizing, loading, and operating margins of the software.

- SIV&V participates in the component Test Readiness Review (TRR) or Flight Readiness Review (FRR), if held in this phase, and reports the results.
- Test beds are used extensively with simulations to ensure that the integrated parts of the system can meet their stated performance requirements. Interfaces are tested exhaustively (e.g. error seeding, stress testing, “what if” scenarios, and the like) to ensure that they are able to support worst case loading and quality criteria.
- Limited amounts of operational testing are performed to ensure that key algorithms and functions meet their deadlines and accuracy requirements. These tests focus on critical functions, Measures Of Performance (MOPs), and Measures Of Effectiveness (MOEs) that are primary factors in the success or failure of future systems test.

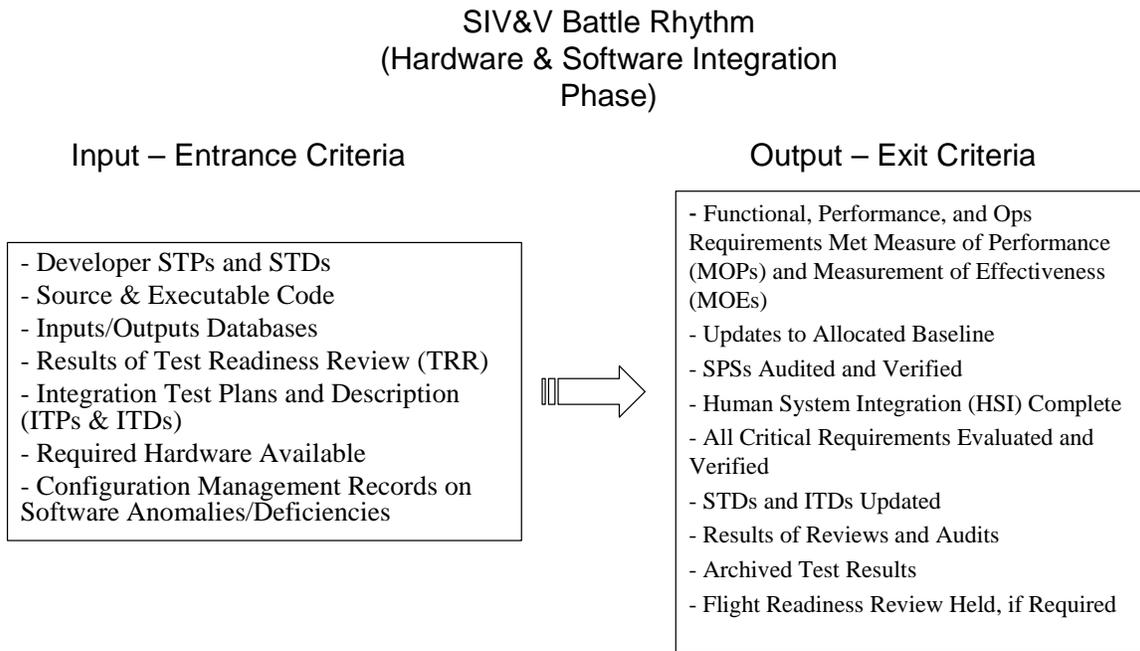


Figure 13. SIV&V Battle Rhythm Hardware and Software Integration Phase (Lewis 1998)

7. Formal Qualification Phase

Validation is the SIV&V activity associated with the developer's formal qualification testing and, in many cases includes flight or live operational certification of the end item. This aspect of validation is often carried forward into the next phase. If pre-qualified hardware already exists, formal validation will be directed only at the software; otherwise, it can involve both hardware and software. However, SIV&V typically focuses on the software aspects. The following tasks are typical of this phase (Lewis 1998):

- SIV&V evaluates qualification test suites for completeness, comprehensiveness, and adequacy. Validation looks back at system and software requirements, determines adequacy of testing, and ensures that mission critical requirements are met.
- SIV&V selects and uses proven analysis tools that support the performance assessment. Then, SIV&V authenticates both the data and test procedures needed for formal qualification and, hence, validation.
- Validation is the formal confirmation or proof that the system meets the user's expectations, can perform its mission to the effectiveness specified by the MOPs and MOEs, and that the software is essentially free of errors and inconsistent behavior that would affect its mission. In a sense, validation is never complete in that every new use or application varies some of the parameters that can affect the outcome or operational integrity of the new system. The objective is to test and validate beyond the nominal test cases (i.e. off-nominal) at a level such that the system does not suddenly fail and can maintain performance even when stressed. Every tester and validator struggles with where to draw the line to declare and affirm that the system is acceptable for use and deployment.
- SIV&V analyzes the scenarios and test cases used for testing and validation to ensure they are sufficient to thoroughly demonstrate the

system capabilities in both nominal and off-nominal (stressing or worst case) situations. Thus, the system is expected to degrade without failing; therefore, the testers and validators have to know where the system begins to degrade and what happens as these limits are exceeded. These behaviors have a great deal to do with whether the system is acceptable for use or not. The software must be designed to accomplish these objectives, and validation provides the essential confirmation.

- SIV&V performs and participates in Logistical Reliability Availability Maintainability (RAM) testing, and effectiveness evaluation, as required by the customer. These are usually run over extended periods of time.
- SIV&V compares test results to other real-world sources.
- SIV&V uses Subject Matter Experts (SMEs) in the technology areas as evaluators.
- SIV&V participates in the Functional and Physical Configuration Audits (FCA & PCA), once administered, and evaluates and confirms that all critical issues have been resolved and closed.
- SIV&V assesses and verifies the training of operators and users, if requested by the customer.
- All software changes occurring during the period are assessed for their impact on source code and base-lined documentation.

SIV&V Battle Rhythm (Formal Qualification Phase)

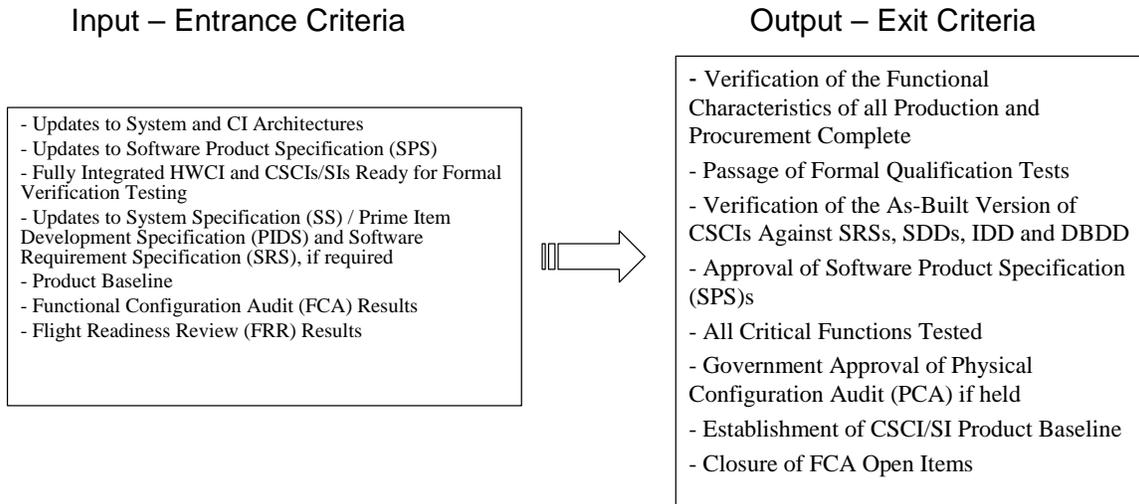


Figure 14. SIV&V Battle Rhythm Formal Qualification Phase (Lewis 1998)

8. Operational Readiness Phase

Operational readiness validation is an important SIV&V phase of system development requiring live fire, flight, or other forms of operational status validation or other forms of pre-operational checkout in a realistic “situational” environment. Essentially, Operational Readiness tests and demonstrations take on the characteristics of the system they support. The following tasks are typical of this phase (Lewis 1998):

- To accomplish this form of validation, SIV&V evaluates the limitations and constraints of the system to satisfy all of the operational requirements and perform all of the necessary tests. SIV&V may work along side the Operational Test and Evaluation (OT&E) personnel in performing these tests.
- SIV&V assesses the “situational” test capabilities (environment, test ranges, test cases, scenarios, targets, behaviors, and so forth) to

ensure the proper mix of testing, the ability to command and control the developed system, and general oversight of the operational testing.

- SIV&V serves as an operational expert for this evaluation together with appropriate tools to support the necessary analysis.
- SIV&V performs a self-assessment based on past and current metrics that enable a quantitative measure of SIV&V effectiveness. SIV&V metrics are often compared to similar development metrics to determine an effectiveness ratio between the subsequent groups.
- All software changes occurring during this phase undergo evaluation for their impact on source code, design, and documentation.
- An operational test often involves Joint Services and Joint Level components where SIV&V assesses interoperability among the various assets that exchange data, status, and operational objectives. SIV&V participates in these tests when possible and ensures that operational objectives meet or exceed the system expectations.

**SIV&V Battle Rhythm
(Operational Readiness Phase)**

Input – Entrance Criteria

- Functional, Allocation and Product Baselines
- Software Product Specification (SPS)s and Version Release Records
- Test-Approved Test Document
- Results of Functional Configuration Audit (FCA) and Physical Configuration Audit (PCA)
- Integration Verification Results
- Results of Flight Readiness Review, if required
- Operational Requirements
- Operational Tests Reports



Output – Exit Criteria

- Results of ORR, if held
- Assessment of Various Operating and Control Modes, States and Behaviors
- Post-Test Analysis Results
- Effectiveness/Performance Analysis Results
- Operational Capabilities Assessment
- Deficiencies/Anomalies Reported and Closed

Figure 15. SIV&V Battle Rhythm Operational Readiness Phase (Lewis 1998)

9. Operations and Maintenance Phase

In the Operations and Maintenance (O&M) phase, SIV&V is typically a scaled-down compressed version of the development and integration phases mentioned previously. The key drivers for O&M SIV&V are Engineering Change Proposals (ECPs) and Pre-Planned Product Improvement (P3I) programs. SIV&V helps to evaluate the impact and magnitude of the proposed changes and determines all the requirements, design, code, documents, and tests that are affected. The following tasks are typical of this phase (Lewis 1998):

- Based on the level and types of changes, SIV&V identifies, selects, and uses an appropriate mix of tools, techniques, and methods to optimize as much as possible the analysis and evaluation of the affected parts of the software and/or system.
- SIV&V is expected to develop a tailored or scaled SIV&V plan, test plan, and cost estimate as appropriate for the size and schedule of the proposed effort.
- SIV&V evaluates the changes for completeness, consistency, correctness, impacts to other parts of the system, and feasibility.
- Typically, SIV&V re-verifies previously base-lined documents and code, examines the support data and documentation such as CASE and analytical tool outputs for completeness, accuracy, correctness, and consistency.
- SIV&V performs testing in accordance with the revised documentation (i.e. Software Test Plans and Test Descriptions).
- SIV&V participates in all design reviews and audits needed to assess and evaluate the changed products at each phase.
- Once the product has evolved to the stage where formal re-qualification is needed, SIV&V participates in the TRR, FCA/PCA, and FRR, as required by the customer.
- All software changes occurring during this period are evaluated for their impacts on source code and base-lined documentation.

- This phase usually covers scaled-down versions of many of the life cycle phases and, therefore, places schedule constraints on all parties: customer, SIV&V, developers, and integrators. However, SIV&V's ability to meet any program constraints or surge capabilities is one of its major strengths.

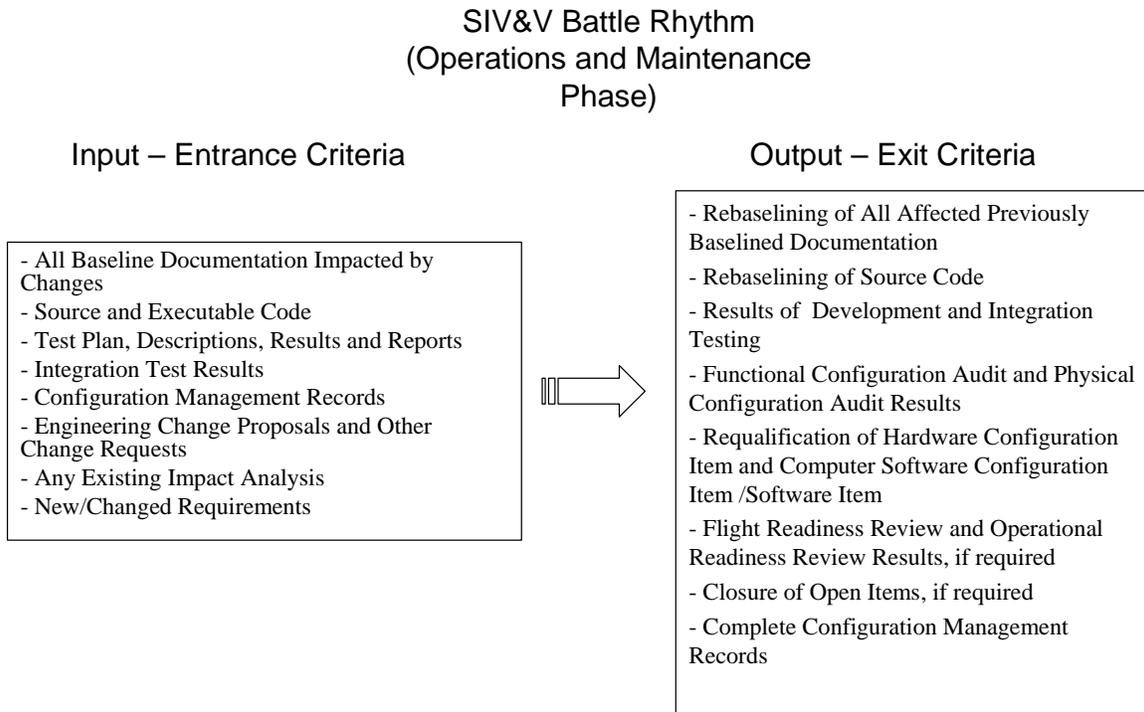


Figure 16. SIV&V Battle Rhythm Operations and Maintenance Phase (Lewis 1998)

D. SIV&V SUPPORTING CASE TOOLS

1. Introduction (I-Logix.com; Telelogics; Spector; Rational; Zambrana)

Computer-Aided Software Engineering Tools (CASE) are supplemental programs that assist in automating software-design and development processes. For example modeling tools, compilers, structure editors, and source-coded systems are forms of CASE tools. CASE tools relieve the programmers from having to work detailed tasks related to hardware, thus allowing them to concentrate on higher-level software system abstractions.

The evolutionary process of CASE tools has progressed to specific tools of software development that assist in defining and validating particular aspects of software system design. Current generations of CASE tools are whole systems within themselves, consisting of multiple tools that assist software teams with designing software systems in a logical progressive pattern.

There are three types of CASE Tools: Design, Hybrids, and Build environment tools. By definition, CASE Design Tools assist diversified cells of engineers with software system specification development. Design Tools further assist engineers in code stubs, documentation, and the automated writing of frameworks that are incorporated into the developer's program or routine. The Unified Modeling Language (UML), initially supported by Grady Booch, Jim Rumbaugh, and Ivar Jacobson, is another language utilized by CASE Design Tools. The development of UMLs has revolutionized the ability of software to produce system specifications that are easily incorporated into a maintainable and productive code. The most promising aspect of CASE Design Tools is that some of them can assist in the design of almost unlimited specifications from document development to embedded systems utilized in military Battle Management software.

CASE Build Tools assist diversified cells of engineers in managing and composing the release of sophisticated software packages; this aids developers in tracking executables, objects, and combinations of sources encompassed within the developed system.

Hybrid Tools are newly developed CASE Tools that combine existing support tools with web services to produce a distributed flexible system capable of managing various styles within software development stages. Hybrid CASE Tools are also capable of incorporating new software enhancements with a minimum effort in labor. "Sourceforce" and "Collab.net" are perfect examples of Hybrid CASE Tools, all of which follow strict integration and design processes. Further, they have the capability to analyze the what, when, and where of the software development process.

The IV&V change agent must acquire the appropriate software tools to evaluate and address each phase of the validation. Developing a large software tool library is useless unless all the technical personnel are highly proficient with the tools and their components.

2. Requirements

Requirements are the compilation of specifications and activities developed by the user or customer in an attempt to lay the guidelines for the architecture of a developing system. In the software community, a holistic approach is used to address requirements. Once the project's requirements are developed and accepted, they are prioritized with respect to maximizing quality, maintainability, and ease-of-tracking of the software system. Therefore, elicitation of software requirements is a key factor in the development of new software systems. Two important factors used to define and develop software requirements are consistency and uniformity, which are essential to reducing system costs (Telelogic).

3. Design

The system software design phase is considered transitional, in that it translates the software designers' concepts, notions and ideas into a semi-structured architecture and resourced entity. The design phase is also iterative in that it develops the allocation of requirements into a specific design. Software design is partitioned into two sub-functions, the system design and the software design. System design is related to hardware development, and software design is related to man-in-the-loop interfaces. Critical resources and requirements are allocated during the software design phase, and questions are answered pertaining to the security of data, maintenance of hardware, software tools and software databases. Examples of software case tools utilized in the design phase are "Rational Rose," "Rhapsody," "McCabe," and "Klockwork" (Spector; I-Logix; Rational; Klocwork).]

4. Code

Coding is defined as translating the user's requirements into a language that allows the computer to execute those requirements. Since "requirements

creep” is a consistent issue, coding must be able to change with requirements. Thus, the logical assumption that requirements are clarified and finalized during the coding process is in fact rarely true for today’s complex systems. Typically, the design requirements are immature as the coding process is started. Given this dilemma, the programmer must try and achieve four goals during the coding process. First, quality, as the most important goal in the coding process, must be emphasized from the beginning of the coding process to assure that quality is incorporated during the maintenance and test phases of software development. Second, uniform and consistent guidelines and processes are required to assure that readable codes and production of logical flows occur. Third, coding documentation must be in a form that is easy to understand and maintain. Programmers dread the maintenance of non-standard codes as extreme efforts are required to maintain these codes. Non-standard codes typically result in negligence of code maintenance. Finally, the documentation is a living document that must be iteratively updated to remain relevant. As the code matures the documentation must become increasingly detailed because the documentation forms the foundation for the operation and maintenance of the software in a cost and time effective manner (Shula; Badeaux).

5. Tracking Database

A software database is an internally developed or Commercial-Off-The-Shelf (COTS) program that tracks errors or bugs from initial to final testing of the software. The basic goals of utilizing a tracking database tool are to find errors, assist in correcting errors, and annotate the corrections using the report function of the database for future reference. The bug-tracking database is a time, cost, and labor saving tool that capitalizes on the tracking and reporting functions of the software. It also enhances the programmer’s abilities to discover, check, and correct errors before the software system enters into the production phase of the life cycle (Shula).

6. Rate Monotonic Analysis (RMA)

Rate Monotonic Analysis (RMA) algorithms were originally developed to analyze the scheduling of periodic tasks to an associated periodic request rate.

The basic function of RMA is to prioritize tasks according to the period in which they occur. In essence, tasks with shorter-time periods are assigned a higher priority, and tasks requiring longer-time periods are assigned a lower priority. In order to implement the RMA certain assumptions are made, for instance, aperiodic tasks in a system routine are considered special, and will displace periodic tasks in the execute routine, or there are consistent run-times for each task, or tasks are not dependent on the completion of other tasks. Task deadlines are consistently designed and noted at the beginning of the next period. Software systems benefit from using RMA by retrieving real-time task conditions and analyzing the results statically to determine whether or not task deadlines are executed, and whether static scheduling test results fall within the boundaries of the system's rare monotonic assumptions (Forman; Klien).

7. Security Assessment (Klocwork; Ghosh; Gilliam; Laliberte)

Software applications that resident on desktops, mainframes, and servers are vulnerable to illegal hacking and attacks. Vulnerabilities in computer software arise from a number of oversights in programming and many times are tracked back to poor software development techniques. Unsecured network links and newly developed methods of attacks contribute to security vulnerabilities. Security Assessment Tools (SAT) utilize current assessment methodologies to identify security risks in resident and networked software. Vulnerabilities are tested in the client's environment, assessed, and prioritized according to the estimated severity of effects. Another approach to assessing security is by exploiting obvious vulnerabilities in break-in attempts, and assessing whether or not the attempts are successful. The routine that allowed the break-in is patched and the security program searches other areas of vulnerabilities. This method is known as penetrate and patch.

Certifying security assessments occur at the component and system level. Assessing and certifying at the component level evaluates whether that component operates as designed in the operating environment without allowing dangerous penetrations. At the system level, software is evaluated at the total

package level for effectiveness against Trojan horses or viruses and finally assessed by SATs and individual evaluations.

Table 7. Recommended SIV&V Tool Suite

TOOL	Requirements	Design	Coding	Tracking Database	RMA	Security Assessment
DOORs	X					
RTM	X					
Rational Rose	X	X				
Rhapsody		X				
McCabe		X	X			
Klockwork		X	X			X
EXCEL Spreadsheets				X		
Flaw Finder						X
Microsoft Access				X		
Timewiz					X	
Doves						X

E. IMPACTS OF ACQUISITION REFORM

Acquisition reform within the Government has led to an increase in the use of performance-based specifications. This approach has some advantages for procurement, but creates great difficulties for SIV&V. The main effect is that the software may be pushed to a lower Work Breakdown Structure (WBS) level that is not visible to the Government. Software and software products do not appear in the upper levels of the WBS. Therefore, they may not be delivered for review, reported on schedules to the Government, or appear in cost breakouts. This

situation makes software tracking, oversight, and SIV&V almost impossible. The response from the contractor concerning this is that the Government is welcome to come to their facility to review data using their electronic systems such as online Software Development Folders (SDFs). Unfortunately, this approach does not work. Getting access to contractor facilities can be very difficult (badges, escorts, and so forth), and obtaining accounts on contractor computer systems can be extremely difficult. These issues are orders of magnitude more difficult, if not impossible, for Government support contractors due to company policies, non-disclosure agreements, proprietary issues, and competition sensitive caveats. Even if access is obtained, locating the needed data without assistance is very difficult since the data is often widely dispersed and poorly organized.

The second issue that is often presented is that since the contractor is a Capability Maturity Model (CMM) Level 3 or higher organization, it is assumed that Government monitoring, access, and even SIV&V of the software is not needed. No data or study supports this position. In fact, the CMM promotes oversight and SIV&V activities. A good Level 3 and above contractor would generate documents, have oversight and tracking, and provide status; however, in the present climate of budget cuts, key processes are often tailored out of a program. Contractor management is fond of saying, "If it is not in the contract, we don't have to do it." Even if the CMM process is followed, Government access may still be limited.

How can this situation be corrected? The simple truth is that if the Government wants something, it must be specifically stated in the contract. A SIV&V contract should be awarded simultaneously with the Prime contract and should continue throughout the life of the system. Document deliverables must be identified, and provisions for providing SIV&V access to early release and draft documents must be included. Schedule and cost reporting requirements must include software elements. And finally, contractor support of SIV&V activities must be indicated. Adding SIV&V into a program late in the development cycle dramatically reduces the effectiveness and ROI of SIV&V.

In a similar vein, contractor proprietary information creates great barriers to SIV&V activities. The contractor indicates that using an internally-developed proprietary product will save the Government money. However, this is rarely the case. Instead, proprietary labels can be used to limit Government visibility into their activities. As a result, documents cannot be obtained, source code cannot be inspected, and other contractors cannot be involved. Often the contractor will reuse one small module within a large new development, yet declare the entire product proprietary. Similarly, a document may contain one paragraph of proprietary information, yet every page of the document is marked proprietary. Non-disclosure agreements are difficult to implement and do not fully solve the problem. Challenging these cases is very difficult since it immediately involves legal groups which are costly in both dollars and schedule. Proper marking of proprietary information and products to the minimum applicable object, and minimizing use of proprietary products can help maximize the ROI on SIV&V. In addition, all proprietary items deemed necessary must be fully defined with supporting rationale during the proposal period. In some areas, the Government should request a non-proprietary solution so that Government ownership and reuse can be obtained. Scrutiny must also occur during the contract's life to make sure that additional proprietary items do not appear. These measures protect both the contractor's and Government's rights to obtain a fair product while allowing SIV&V to occur to the maximum extent possible.

F. KEYS TO SUCCESSFUL SIV&V

To maximize ROI, the following key elements, that are lessons learned from SED participation in multiple software development and software SIV&V efforts during the past decade, should be utilized.

- 1) Government Program Managers (PMs) should plan for SIV&V to be included in the initial phases of a new development program in order to properly plan for and execute a comprehensive SIV&V effort. A significant phase of SIV&V occurs during the requirements generation process, from which SIV&V ensures that the necessary system requirements/capabilities are established to

quantify the desired performance of the new system. Many uninformed PMs mistakenly assume that SIV&V does not need to be included until the software implementation is complete, thereby limiting SIV&V to only a software test role. By being involved early in the software development cycle, SIV&V is more knowledgeable about the software products (documents, requirements, and the like) and capabilities of the system, which will result in a more thorough SIV&V effort, particularly during software testing.

2) Software developed under capabilities-based program acquisition strategies are more difficult to verify and validate. The program emphasis on capabilities, rather than requirements, results in the SIV&V organization delineating between the necessary software, hardware, and personnel capabilities as part of the overall system's capability. Most system capabilities require a system-level test or analysis in order to verify and validate, which implies that the SIV&V organization should have access to the overall system assets and/or system data. Such access is usually more difficult to obtain from a cost and schedule perspective.

3) SIV&V funding should be managed separately from the organization managing the software development and other software support functions. Such separation helps ensure that the SIV&V effort does not suffer financially from software budget reductions and/or cost overruns by the Prime Contractor.

4) The Integrated Product Team (IPT) concept can make SIV&V difficult. If SIV&V is properly included as part of the IPT, and the IPT lead is the Prime Contractor, then the SIV&V organization has due allegiance to the Prime Contractor, in addition to the Government. Relationships between the Government, SIV&V, and the Prime Contractor need to be partnered and agreed upon prior to the IPT launch.

5) The SIV&V organization must be given necessary access to the software-related materials (code, data, documentation, and the like) required to perform SIV&V. The Government must ensure that the Prime Contractor agrees

with such access. It is best if this agreement is established in a written document such as the Prime Contractor's Scope of Work (SOW) or Software Development Plan (SDP). The Government should be prepared to compensate the Prime Contractor for any personnel support associated with responding to the SIV&V's requests for information. This support should not be an additional cost; it should be included as part of the Prime Contractor's contract within the SOW.

6) The use of the SEI CMM does not eliminate the need for SIV&V. A Software Development Organization (SDO) can have a high CMM rating but still experience software problems. Some SDOs may choose to sidestep approved processes and procedures in an attempt to cut costs or save schedule. The SIV&V organization is present to ensure that:

- The software is being built right.
- The right software is being built.

7) The SIV&V contract should be awarded simultaneously with the Prime contract and should continue through the life of the system. SIV&V involvement should begin at the beginning of the requirements stage so that SIV&V can impact the maximum number of requirement errors during the requirements phase when the cost savings are greatest. In addition, SIV&V subject matter experts are developed concurrently with developer experts. Therefore, the customer has two sources of expertise. Also, the developer can use the SIV&V team as a source of expertise during times of personnel turn-over and growth, and when it is more efficient to rely on SIV&V expertise rather than impact the development schedule. This also positions the SIV&V team to continue functioning as SIV&V through Post-Deployment Software Support (PDSS) or to "be handed" the system for maintenance once it no longer becomes cost-effective to the developer.

8) The SIV&V team should report directly to the same Government Point of Contact (POC) as the developer. This ensures that the Government can manage the relationship between the developer and SIV&V team, and make certain that the relationship is supportive rather than adversarial. In addition, the

Government customer has direct access to all sources of information, which can be helpful in times of disparity. This also enforces the independence of SIV&V.

9) SIV&V must be built into the contract vehicle of the developer. The contract should provide for SIV&V to happen “in-phase” or concurrently. Draft documents and build deliverables should be made available to the SIV&V team. Also, SIV&V should be involved during the development and testing cycle. This allows errors to be found as quickly and efficiently as possible, and corrected in-phase as often as possible, which dramatically decreases the cost to fix findings. “In-phase” SIV&V also removes some of the adversarial relationship between the developer and SIV&V because there is less cost and schedule penalty for errors found, since they are more likely to be addressed in the earliest phase possible. A close relationship of trust and respect must be established for the SIV&V team to be involved at this level. The SIV&V team should be viewed as a reliable and cost-effective resource by the developer.

10) The contract should state that source code will be available to the SIV&V team. Source code allows the SIV&V team to inspect algorithms and establish test cases that might otherwise go un-inspected or un-tested. It also allows the SIV&V team to test hard-coded values and make use of commercial source analysis tools such as McCabe and VectorCast.

11) SIV&V should be given a scope outside of “classical” SIV&V. The “independence” in SIV&V gives it a natural tendency to excel in areas such as, safety analysis, “what if” test scenarios, and case studies. The SIV&V team should use the developer’s tools to ensure “apples-to-apples” results, but also should use or develop test and analysis tools outside of those used by the developer. The SIV&V team should be scoped to generate test cases to stress the system in unique and original ways. The SIV&V team should also be scoped to analyze test data using all resources available to spot as many anomalies as possible.

12) Often, software development requirements are considered too late in the System Development process. System engineering, historically, has focused on hardware capability and availability. Hardware requirements and design typically drive the System Development effort and software often ends up as an afterthought that must operate with given hardware restrictions. Software is often considered not to be a critical component of System Engineering. Systems that are considered to be software-intensive typically have limited staffing (less than 10% of the acquiring organization) to support the system software acquisition. System Engineering should have both a hardware and a software engineering focus. If software engineering is not accounted for until late in the overall System Development process, the software development is reactionary to the hardware development, which is often counterproductive.

13) SIV&V should include software security assurance as part of its scope. Due to the constantly increasing threat of software intrusions, the SIV&V effort should analyze the subject software to identify and help eliminate any potential software security vulnerabilities or weaknesses. DoD Directive 8570 emphasizes the importance of information security and places the responsibility of software security at the respective Product Manager level.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SUMMARY, RECOMMENDATIONS, AND CONCLUSIONS

A. SUMMARY

Over the past 60 years, software has become the most critical component of not only every major weapons system employed by the United States Military, but also serves as the very foundation for the commercial infrastructure (e.g. financial, transportation, agricultural, utilities, medical, and the like) that we as a society have come to depend upon as we go about our daily lives. Countless examples, some of which have been presented in this thesis, of software failures and the resulting consequences, in lost time, resources, and regrettably sometimes even lives, abound in today's world. In this respect, SIV&V can be viewed as "cheap insurance" against the prospect of catastrophic failures of fielded software that could possibly have tragic results.

This thesis strives to address this need for better quality, highly reliable software, by providing the reader with the rationale, guidance, lessons learned, and the tools necessary to solve these critical and complex software dilemmas. By utilizing the information provided in this thesis, Government Program Managers and their commercial counterparts can significantly improve the odds of fielding successful high-quality reliable software products. It is a process where by the user can answer the key questions "Are we building the right thing?" and "Are we building the thing right?" Thus, government users can be assured with a high level of certainty that their weapons systems will not fail them during a critical operational moment, and commercial users can rest assured that the very infrastructure that they have come to depend on to run our modern society will not collapse.

B. CONCLUSIONS

It is essential that the government agency or commercial entity and their software engineering support be administratively and technically competent to effectively implement a quality software IV&V process. Faults and errors made by software engineering support cause unnecessary expenditures of time and

money. The mishandling of these funds results in an enmity towards the organization that is damaging both politically and institutionally as it affects the ability of the organization to complete the project. Beginning the project with the correct complement of skills for the task precludes errors, and ensures significant system/software errors are surfaced early (Makowsky).

An in-depth analysis has been conducted on why it is important to conduct Software Independent Verification and Validation. Within this thesis, the team has documented examples of DoD and non-DoD uses and non uses of SIV&V. Our research solidifies why it is important to use SI&V prior to deployment of a new system or product. The Software Engineering Directorate (SED), at the U.S. Army Aviation and Missile Command (AMCOM) has extensive knowledge in reviewing and evaluating software development, conducting design reviews, and software integration and testing for software builds. The example set by SED can serve as a model for both government and commercial entities that are determined to improve their software engineering processes through utilization of SIV&V.

Finally, this thesis serves as a starting point and tutorial for implementation of the SIV&V process. It provides a listing of suggested sources of policy and guidance, suggested software computer CASE tools, life cycle phased descriptions of methods and criteria that can be utilized, and lessons learned from other programs. It is hoped that the reader will find the information within these pages to be helpful and inspiring in their pursuit of software excellence.

C. RECOMMENDATIONS

Software development and SIV&V capabilities have increased significantly during the past decade due to the expanding role of software in systems and advances in tools and processes. Software development productivity has increased due to the availability of integrated Computer-Aided Software

Engineering (CASE) toolsets that provide almost seamless transition between the requirements, design, implementation, and test phases of the development cycle.

The Software Engineering Directorate (SED), as a Government agency, possesses a cadre of CASE tools and has an impressive advantage as a SIV&V resource. SED is a CMM Level IV organization for software development and is implementing the practices of Team Software Process (TSP), Personal Software Process (PSP), and Capability Maturity Model – Integrated (CMMI) Level IV. Therefore, SED's development expertise lends itself naturally to SIV&V practices. Using SED as the SIV&V team within the Army specifically, and DoD in general, minimizes conflicts of interest and proprietary issues, and maximizes independence. In addition, SED is trusted by the Army Evaluation Center (AEC) as a resource to perform higher levels of testing, resulting in significant savings to the Army. Many Army systems have resources located in-house, which make them available for interoperability testing using both internal and external Army and DoD network resources. This gives the customer access to an expanded tool set.

SED is also a cost-effective source of SIV&V as its customers can leverage cost reductions through existing contract vehicles which utilize onsite pricing structures and efficient operations. SED is the Life Cycle Center for the ARMY and SIV&V, and lends itself to the development of experts that can maintain a system throughout its life cycle.

Each military service should consider establishing a "Center of Excellence" for SIV&V. These centers would focus on providing SIV&V for each of the services programs. SED has a broad base of SIV&V experience and capability that can be made available to other programs and services. NASA has established such a SIV&V facility in West Virginia. This facility provides SIV&V support to all NASA centers utilizing both Government and contractor personnel. This concept of SIV&V "Centers of Excellence" can be extended to the commercial software industry as well. Similar to the engineering and standards

societies that have founded to support various industries like IEEE, ANSI and the like, these SIV&V “Centers of Excellence” could be organized by industry and/or criticality of function or service that the software is providing. Examples of “Centers of Excellence” here would be medical software, aviation software, nuclear power software, or financial software.

A follow-on concept definition study should be performed to make recommendations regarding the organization, management, and execution of the proposed SIV&V “Centers of Excellence.”

D. ANSWERS TO RESEARCH QUESTIONS

1. Primary Research Question

What are the benefits of and rationale for PMOs and others for using SIV&V?

See Chapter I Sections B and G, Chapter II Sections A, C1, and C3, and Chapter III Section B.

2. Secondary Research Questions

What software CASE tools are available for software V&V?

See Chapter III, Section D, for a listing of SIV&V CASE Tools.

What key things should be done or considered when conducting SIV&V?

When conducting SIV&V it is important to do or consider the following key things:

- Document the organizational structures and processes that support centralized coordination of SIV&V activities and deliverables to achieve maximum efficiency.
- Standardize SIV&V policies, processes, and products across the program.
- Identify technical issues and trends in a timely manner for management focus.
- Integrate and enhance existing SIV&V activities within the SIV&V community.

- Provide leadership insight into SIV&V plans, milestones, and progress.
- Consider how SIV&V will reduce risk of program, element or component failure due to operational baseline or simulation software defect(s).
- Develop and test software to ensure high confidence in the system and component capabilities.
- Ensure software is mature and dependable.
- Ensure SIV&V is executed via the coordinated efforts of all program directorates and component project offices.
- Ensure SIV&V activities examine the Prime contractor's software development plans, processes, and products throughout the software development lifecycle.
- Ensure early identification of products for delivery, otherwise, your support may not provide any added benefit.
- Identify coordination requirements and information flow for access to software, documentation, and data items.
- Ensure early identification of tracking mechanisms and tools as well as establishing configuration status accounting processes.

What are the SIV&V process steps?

See Chapter III, Applying SIV&V In The Cycle Phase.

Another excellent source is the SEES process. The U. S. Army Aviation and Missile Command's Missile Research, Development, and Engineering Center (AMRDEC), Software Engineering Directorate (SED), has developed the Software Engineering Evaluation System (SEES). The SEES defines the Independent Verification and Validation (IV&V) tasks, including procedures for crucial unique software development issues that may be performed by the SED in support of a Program Management Office (PMO) request to evaluate software intensive systems. The overall SEES approach, based upon DOD-STD-2167A [1], is depicted on the Integrated System Diagram (ISD) provided in APPENDIX

A. The SEES utilizes analysis methods and practices compatible with the developer's effort. The results of the SEES methods and practices document the software engineering accuracy, as well as the deficiencies of the developmental products.

How has acquisition reform affected SIV&V?

See Chapter III, Section E.

What lessons have been learned from past programs that have utilized SIV&V?

In most cases, the SIV&V agent tries to *maintain their independence* while staying intimately involved in the day-to-day software activities performed by the prime developer. If this were not the case, the government assessments would be limited to the developer's test program only. This concept is a big concern because the software will be executed in ways that the rule-writers do not fully cover. Accordingly, the developer decides what, when, where, and how the software will be tested. The developer also decides what and when test data will be available for government review. Simply put the SIV&V agent provides oversight for the developer's test plans, procedures, testing, and data analysis.

Another short fall or lesson learned is that if SIV&V is not performed then the Government will have no software risk mitigation capabilities for defect identification and prevention. This means the government cannot independently address high-risk areas (interfaces, critical algorithms, and the like), perform any accelerated software checkout activities or conduct additional regression testing, or respond to other element IV&V test concerns or findings.

Yet another lesson learned is that there are large risks associated with utilizing a reduced SIV&V or tailored team. The consequences of this are as follows:

- Software requirements become program risk items, as there is no Government insight into missing or incomplete requirements, traceability or flow down, or requirements certification.

- Software design processes become questionable, as there is no Government contribution to the software design documentation, and no subject matter experts for software design and implementation
- Software Testing and integration become high-risk items, as there is no “on-the-ground” insight or test witnessing (regarding status) of FQTs, internal integration events, or field software check-outs. This is why SIV&V is declared “cheap insurance” for the project offices, as they are solely dependent upon SIV&V for solutions to software problems.

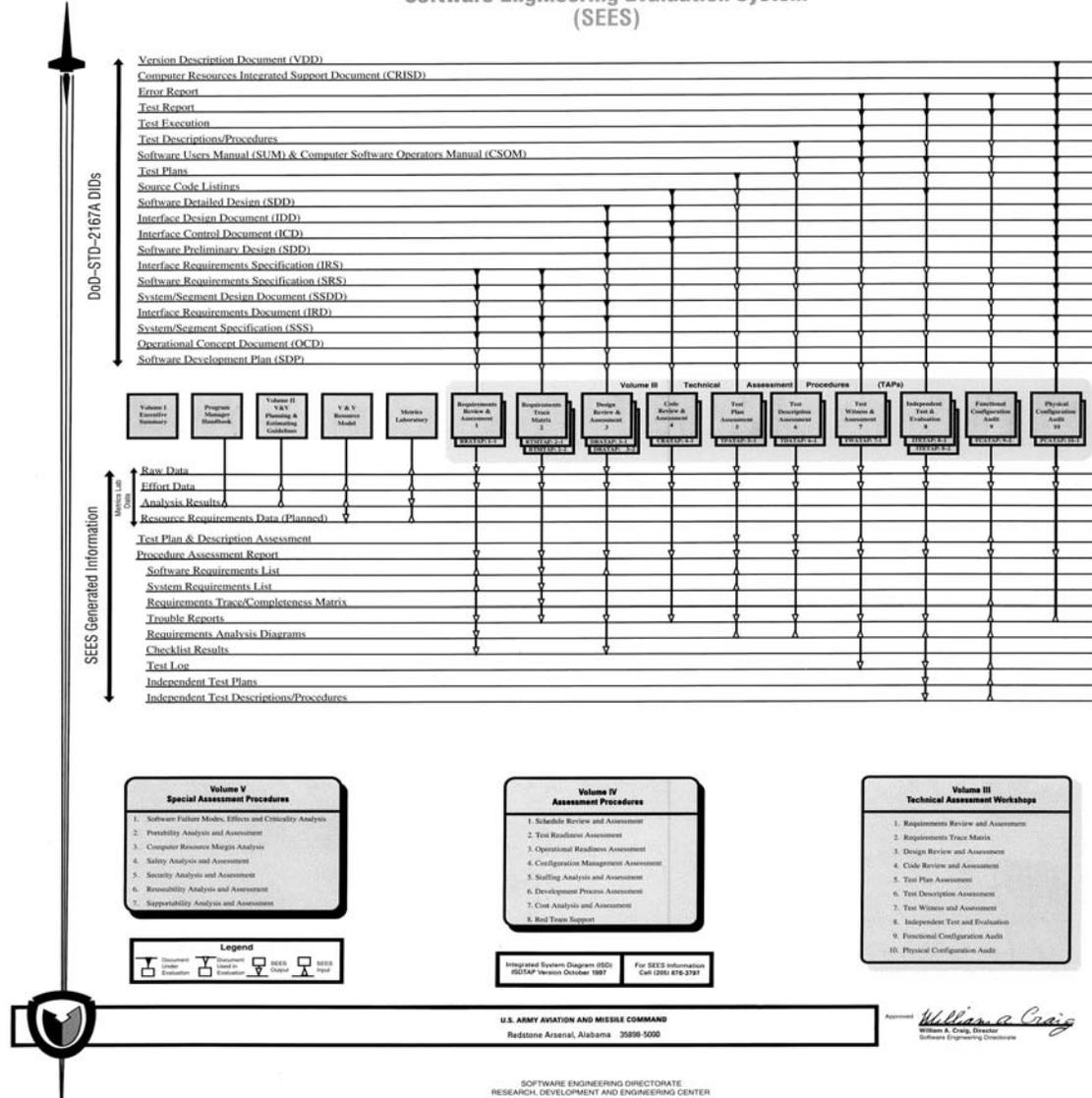
E. RECOMMENDATIONS FOR FURTHER STUDY

Good software should be cost effective, reliable, maintainable, defect free and above all usable. Presently software possess very few of these attributes. Software is simply dreadful today and is becoming shoddier all the time. Industry and the government need to partner to find better ways to educate people both within government and industry on the necessity of SIV&V and its use. Accordingly, the software community should continue to develop and evolve the SIV&V process and tools to keep pace with software evolution and its increasing complexities. One such area that should be considered for further study is called Independent Integrated Verification and Validation (I^2V^2) (Dalrymple). I^2V^2 is an evolution of IV&V that suggests that SIV&V agents should become an integral part of the acquisition process by becoming involved at the very beginning of the development process. How this would be accomplished and how the SIV&V agents would retain their objectivity and independence are key questions that need to be answered. Failure of the software industry to rise to this challenge in conjunction with the increasingly litigious nature of our society, may very well lead to expensive litigation as lawyers and special interest groups discover yet another lucrative commercial enterprise ripe for exploitation.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX: INTEGRATED SYSTEM DIAGRAM (ISD)

Software Engineering Directorate Software Engineering Evaluation System (SEES)



THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- “AT&T Wireless Blames Software Glitch for Losses.” *Consumer Affairs Inc.* 23 Jan. 2004. 16 Nov. 2006
<http://consumeraffairs.com/news04/attw_glitch.html>.
- Atwood, Jeff. “The Long, Dismal History of Software Project Failure.” *CodingHorror.com*. 15 May 2006. 12 Sep. 2006
<<http://www.codinghorror.com/blog/archives/000588.html>>.
- Badeaux, Christie. “Collab Software Coding Standards Guide for Java.” 30 July 2001. 22 Sep. 2006
<<http://collaboratory.emsl.pnl.gov/docs/collab/sam/CodeStandards.html>>.
- Barber, G., J. B. Dabney, and D. Ohi. “Estimating Direct Return on Investments of Independent Verification and Validation.” Titan Systems Corp. NASA SIV&V Facility, Fairmont, WV and Department of Systems Engineering, University of Houston - Clear Lake, Houston, TX.
- Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.
- Boehm, Barry W. *Software Risk Management*. Washington, DC: IEEE Computer Society Press, 1989.
- “Building a better bug-trap.” *Economist.com*. 19 Jun. 2003. 3 Dec. 2006
<http://www.economist.com/displaystory.cfm?story_id=1841081>.
- Callahan, Jack. *Verification and Validation: An Editorial Primer*. 1997. 19 Nov. 2006 <<http://www.icse-conferences.org/1997/news/callahan.html>>.
- Dalrymple, Edgar and Mike Edwards. Conference Briefing. *Independent Integrated Verification and Validation I²V²*. Conference on the Acquisition of Software-Intensive Systems. 28-30 Jan. 2003. 1 Nov. 2006
<<http://www.sei.cmu.edu/programs/acquisition-support/conf/2003-presentations/edwards.pdf>>.
- Defense Acquisition University. *Advanced Systems Planning, Research, Development and Engineering (SYS 301) Volume 1*. Fort Belvoir, VA: US Government Printing Office, 2003.

Department of the Air Force, Software Technology Support Center. *Guidelines for Successful Acquisition and Management (GSAM) of Software-Intensive Systems: Weapon Systems Command and Control Systems Management Information Systems*. Version 3.0 May 2000.

“Entries from the Software Failure Hall of Shame, Part 1.” *g2zero.com*. 6 Jul. 2006. 16 Nov. 2006
<http://www.g2zero.com/2006/07/notable_entries_from_the_softw_1.html>.

“Food For Thought: What is Software Quality Assurance?” *Software Quality Consulting, Inc.* Jan. 2005, Volume 2 Number 1. 22 Oct. 2006
<<http://www.swqual.com/newsletter/vol2/no1/vol2no1.html>>.

Forman, Nate. Class Lecture. *Rate Monotonic Theory*. University of Texas, EE382C Class. 20 Mar 2000. 3 Dec. 2006
<www.ece.utexas.edu/~bevans/courses/ee382c/lectures>.

Ghosh, Anup K. and Gary McGraw. “An Approach for Certifying Security in Software Components.” *CIGITAL.com*. 3 May 1998. 8 Sep. 2006
<<http://www.cigital.com/papers/download/cert.pdf>>.

Gilliam, David P., John C. Kelly, John D. Powell, and Matt Bishop. “Development of a Software Security Assessment Instrument to Reduce Software Security Risk.” *Proceedings of the 10th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (2001)*: 144-149. Cambridge, MA, 20-22 Jun. 2001. 23 Sep. 2006
<<http://nobs.cs.uc.david.edu>> or
<<http://csdl2.computer.org/persagen/DLabsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/wetice/2001/1269/00/1269toc.xml&DOI=10.1109/ENABL.2001.953404>>.

“I-Logix Telelogic RHAPSODY.” *ilogix.com*. 2006. 10 Jun. 2006
<<http://www.ilogix.com/sublevel.aspx?id=284>>.

Klein, Mark. “Rate Monotonic Analysis, Software Technology Roadmap.” *Carnegie Mellon University*. 10 January 1997. Software Engineering Institute. 25 Aug. 2006
<http://www.sei.cmu.edu/str/descriptions/rma_body.html>.

“KLOCWORK K7 Products.” *klocwork.com*. 2006. 22 Sep. 2006
<<http://www.klocwork.com/products/klocworkk7.asp>>.

Knutson, Charles, and Sam Carmichael. “Safety First: Avoiding Software Mishaps.” *Embedded Systems Programming Magazine*. 16 Nov. 2006
<<http://www.embedded.com/2000/0011/0011feat1.htm>>.

- Laliberte, Scott. "Security Assessment and Risk Analysis Laboratory." *U.S. Army Communications-Electronics Life Cycle Management Command, Software Engineering Center*. 9 Sep. 2006
<http://www.sec.army.mil/secweb/facilities_labs/sara_lab.html>.
- Lewis, R.O. *Independent Verification and Validation (IV&V) Process Chart*. Huntsville, AL: TEC Masters, Inc., 1998.
- Lewis, R.O. *Independent Verification & Validation: A Life Cycle Engineering Process for Quality Software*. New York: John Wiley & Sons, 1992.
- Makowsky, Lawrence C. *A Guide to Independent Verification and Validation of Computer Software* (Technical Report, USA-BRDEC-TR//2516). Fort Belvoir, VA: U. S. Army, Belvoir Research, Development and Engineering Center, Jun. 1992.
- Mann, Charles C. "Why Software Is So Bad." *Technology Review*. July/Aug 2002: 33-38. 3 Dec. 2006
<<http://moosehead.cis.umassd.edu/cis580/readings/WhySoftwareIsSoBad.pdf>>.
- Missile Defense Agency, Ground Based Mid-Course Defense. Briefing. *PrelimDesi_January_10_03 IVV Overview1*. 10 Jan. 2003.
- National Aeronautics and Space Administration (1985). *The Cost Effectiveness of Software Independent Verification and Validation* (NASA RTOP #323-51-72). Pasadena, CA: Jet Propulsion Laboratory.
- National Aeronautics and Space Administration (2001). *NASA Policy Directive 8730.4*. 3 Dec. 2006
<<http://www.hq.nasa.gov/office/codeq/doctree/87304.htm>>.
- Persons, Warren L., and Lawrence, J. Dennis. "Class 1E Software Verification and Validation: Past, Present, and Future" (Technical Report UCRL-JC-114806). *21st Water Reactor Safety Information Meeting* (1993). Bethesda, MD, 25-27 Oct. 1993. 22 Oct. 2006
<http://www.osti.gov/energycitations/product.biblio.jsp?osti_id=10128966>.
- "Rational ROSE XDE Developer Plus." *IBM.com*. 2006. 10 Jun. 2006
<<http://www-306.ibm.com/software/awdtools/developer/plus/features/>>.
- Reiss, Steven P. *A Practical Introduction to Software Design with C++*. New York: John Wiley & Sons, 1998. 17 Oct. 2006
<<http://cs.brown.edu/courses/cs190/2006/reiss/chap15.pdf>>.

- Robot, Cornelis. "Introduction to Software History." *The History of Computing Project*. 17 Oct. 2006
 <http://www.thocp.net/software/software_reference/introduction_to_software_history.htm>.
- Rogers, Richard A.Dr., Dan McCaugherty, and Dr. Fred Martin. Case Study Briefing. *A Case Study of IV&V Return on Investment (ROI)*. Titan Systems Corporation. 2001. 3 Dec. 2006
 <<http://www.dtic.mil/ndia/systems/Rogers2.pdf>>.
- Rombach, Dieter. Lecture. *Software Engineering I (WS 03/04)*. Technische Universität Kaiserslautern Fachbereich Informatik AG Software Engineering. Kaiserslautern, Germany, 27 Oct. 2003. 17 Oct. 2006
 <<http://www.agse.informatik.uni-kl.de/teaching/se1/ws2003/notes/Chapter1.pdf>>.
- Rosenberg, Linda. Briefing. *IV&V at NASA*. Software Engineering Workshop. 30 Nov. 2000. 16 Nov. 2006
 <http://sel.gsfc.nasa.gov/website/sew/2000/topics/CharlieVanek_Slides.PDF>.
- Shridhar, Surbhi. "V&V – Verity and Value." *Cybermedia India Online Limited*. 13 Apr. 2004. 22 Oct. 2006
 <<http://www.ciol.com/content/search/showarticle1.asp?artid=56358>>.
- Shula, Shilpa V., and David F. Redmiles. "Collaborative Learning in a Software Bug-Tracking Scenario." Irvine, CA: University of California, Irvine, 16 Oct. 1996. 22 Sep. 2006 <<http://www.ics.uci.edu/~redmiles/publications/C018-SR96.pdf>>.
- Slabodkin, Gregory. "Software Glitches Leave Navy Smart Ship Dead in the Water." *Government Computer News*. 13 Jul. 1998. 16 Nov. 2006
 <http://www.gcn.com/print/17_17/33727-1.html>.
- Spector, David H. M. "Case Tools." *The O'Reilly Network*. 1 Aug. 2002. 3 Dec. 2006 <<http://www.zeitgeist.com/Articles/CaseTools.pdf>>.
- "Telelogic DOORS." *AmericasNetwork.com*. 2006. 16 Sep. 2006
 <http://whitepapers.americasnetwork.com/detail/PROD/1108021988_307.html>.
- Tran, Eushuan. "Verification/Validation/Certification." *Topics in Dependable Embedded Systems*. 1989. Carnegie Mellon University. 17 Oct. 2006
 <http://www.ece.cmu.edu/~koopman/des_s99/verification/index.html>.

U.S. Army, Research, Development, and Engineering Command (RDECOM), Aviation and Missile Research, Development and Engineering Center (AMRDEC), Software Engineering Directorate (SED). Briefing. *TTEC Tailored SIVV Strategy Updated Final*. Redstone Arsenal, AL, 5 May 2003.

Walters, Dan F. "Writing an Effective IV&V Plan." *CrossTalk The Journal of Defense Software Engineering*. Nov. 2000. 19 Nov. 2006 <<http://www.stsc.hill.af.mil/CrossTalk/2000/11/walters.html>>.

Womack, James P., Daniel T. Jones, and Daniel Roos. *The Machine that Changed the World*. New York, NY: Harper Collins, 1991.

Zambrana, Michael and Dennis Singer. *Space and Missile Systems Center (SMC) Software Acquisition Handbook Version 1.0*. U.S. Air Force, 9 Feb. 2004. 3 Dec. 2006 <<http://ax.losangeles.af.mil/axl/sacqhdbk.pdf>>.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Defense Logistics Studies Information Center
U.S. Army Logistics Management Center
Fort Lee, Virginia
4. OASA (RDA)
ATTN: SARD-ZAC
Washington, DC
5. Professor Brad Naegle..
Naval Post Graduate School
Monterey, California
6. Dr. Bill Craig
Software Engineering Directorate (SED)
AMSRD-AMR-BA
Redstone Arsenal, Alabama
7. William A. Mathis
Huntsville, Alabama
8. Alexis P. von Spakovsky
Huntsville, Alabama
9. David PattersonCommander:
Redstone Arsenal ATTN: David L. Patterson
BLDG 5250 SFAE-MSLS-PF-LD-SD-FS
Redstone Arsenal, Alabama
10. Reffela Davidson
Huntsville, Alabama