Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

DEPARTMENT OF DEFENSE

JOINT SOFTWARE SYSTEMS SAFETY ENGINEERING HANDBOOK

DEVELOPED BY THE JOINT SOFTWARE SYSTEMS SAFETY ENGINEERING WORKGROUP

Original published December 1999
Version 1.0 Published August 27, 2010

Naval Ordnance Safety and Security Activity
3817 Strauss Avenue, Building D-323
Indian Head, MD 20640-5555

Prepared for:
Director of Defense Research and Engineering

Distribution Statement A
Approved for public release; distribution is unlimited.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table of Contents

1 Overview ... 1
2 Introduction to the Handbook ... 3
2.1 Introduction ... 3
2.2 Purpose ... 4
2.3 Scope ... 4
2.4 Authority and Standards ... 5
2.5 Handbook Overview ... 6
2.5.1 Historical Background ... 6
2.5.2 Management Responsibilities ... 7
2.5.3 Introduction to the Systems Approach ... 7
2.5.3.1 The Hardware Development Lifecycle ... 8
2.5.3.2 The Software Development Lifecycle ... 9
2.5.3.2.1 Grand Design and Waterfall Lifecycle Model ... 10
2.5.3.2.2 Modified V Lifecycle Model ... 12
2.5.3.2.3 Spiral Lifecycle Model ... 13
2.5.3.2.4 Object-Oriented Analysis and Design ... 16
2.5.3.2.5 Component-Oriented and Package-Oriented Design ... 16
2.5.3.2.6 Extreme Programming ... 17
2.5.3.3 The Integration of Hardware and Software Lifecycles ... 18
2.5.4 A Team Solution ... 18
2.5.5 Systems of Systems Hazards and Causal Factors ... 20
2.5.5.1 Safety as a System Property ... 20
2.5.5.2 Functional Hazard Causal Factors ... 20
2.5.5.3 Interface-Related Hazard Causal Factors ... 21
2.5.5.4 Zonal Hazard Causes ... 21
2.5.5.5 Data Interfaces ... 22
2.5.5.6 COTS ... 23
2.5.5.7 Technology Issues ... 23
2.6 Handbook Organization ... 23
2.6.1 Planning and Management ... 25
2.6.2 Task Implementation ... 26
2.6.3 Residual Safety Risk Assessment and Acceptance ... 26
2.6.4 Supplementary Appendices ... 26
3 Introduction to Risk Management and System Safety ... 27
3.1 Introduction ... 27
3.2 A Discussion of Risk ... 27
3.2.1 Risk Perspectives ... 28
3.2.2 Safety Management Risk Review ... 28
3.3 Types of Risk ... 29
3.4 Areas of Program Risk ... 30
3.4.1 Schedule Risk ... 31
3.4.2 Budget Risk ... 33
3.4.3 Sociopolitical Risk ... 33

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

3.4.4 Technical Risk 34
3.5 System Safety Engineering 35
3.6 Safety Risk Management 38
3.6.1 Initial Safety Risk Assessment 39
3.6.1.1 Mishap, Hazard, and Failure Mode Identification 39
3.6.1.2 Severity Categories 40
3.6.1.3 Probability Levels 41
3.6.1.4 Mishap Risk Index 42
3.6.2 Safety Order of Precedence 44
3.6.3 Elimination or Risk Reduction 44
3.6.4 Quantification of Residual Safety Risk 46
3.6.5 Managing and Assuming Residual Safety Risk 47
4 Software System Safety Engineering 48
4.1 Introduction 48
4.1.1 Section 4 Format 50
4.1.2 Process Charts 51
4.1.3 Software Safety Engineering Products 52
4.2 Software Safety Planning and Management 52
4.2.1 Planning 54
4.2.1.1 Establish the System Safety Program 59
4.2.1.2 Defining Acceptable Levels of Risk 60
4.2.1.3 Planning for Two Distinct Processes 60
4.2.1.3.1 Software Safety Assurance and Integrity Process 61
4.2.1.3.2 Software Safety Hazard Analysis Process 63
4.2.1.4 Defining and Using the Software Criticality Matrix 64
4.2.1.5 Defining the Requirements for Level of Rigor 69
4.2.1.6 Program Interfaces 73
4.2.1.6.1 Management Interfaces 75
4.2.1.6.2 Technical Interfaces 76
4.2.1.6.3 Contractual Interfaces 77
4.2.1.7 Contract Deliverables 78
4.2.1.8 Development of the Mishap Risk Index 79
4.2.1.8.1 Mishap Severity 81
4.2.1.8.2 Mishap Probability 81
4.2.2 Managing the Software Safety Program 83
4.3 Software Safety Task Implementation 87
4.3.1 Analyze and Comprehend the Conceptual Design Baseline of the System 92
4.3.2 Define Software Assurance Levels of Rigor 92
4.3.3 Functional Hazard Analysis 93
4.3.4 Preliminary Hazard Analysis 96
4.3.4.1 PHL Development 96
4.3.4.2 PHA Development 98
4.3.5 Safety Requirements Analysis 103
4.3.5.1 Categories of Software Safety Requirements 106
4.3.5.1.1 Contributing Software Safety Requirements 106
4.3.5.1.2 Generic Software Safety Requirements 106

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.3.5.1.3 Mitigating Software Safety Requirements 109
4.3.5.2 Derive System Software Safety Requirements in the SRA 110
4.3.5.2.1 Preliminary SSRs 110
4.3.5.2.2 Matured SSRs 111
4.3.5.3 Documenting SSRs 112
4.3.5.4 Software Analysis Folders 113
4.3.6 Preliminary Software Design, SSHA 113
4.3.6.1 Attributes of Fault Management 117
4.3.6.2 Other Design Considerations for Safety 118
4.3.6.3 Example Techniques of Preliminary Software Design Analysis 119
4.3.6.3.1 Module Safety-Criticality Analysis 119
4.3.6.3.2 Program Structure Analysis 120
4.3.6.3.3 Traceability Analysis 122
4.3.7 Detailed Software Design, SSHA 122
4.3.7.1 Participate in Software Design Maturation 123
4.3.7.2 Detailed Design Software Safety Analysis 125
4.3.7.2.1 Safety Interlocks 127
4.3.7.2.1.1 Checks and Flags 127
4.3.7.2.1.2 Firewalls 127
4.3.7.2.1.3 Come-From Programming 128
4.3.7.2.1.4 Bit Combinations 128
4.3.7.2.2 “What If” Analysis 128
4.3.7.2.3 Safety-Critical Path Analysis, Thread Analysis, and UML Sequence Diagrams 129
4.3.7.2.4 Identifying Potential Hazard Causes Related to Interfacing Systems 129
4.3.7.3 Detailed Design Analysis Related Sub-Processes 130
4.3.7.3.1 Process Flow Diagram Development 130
4.3.7.3.2 Code-Level Analysis 130
4.3.7.3.2.1 Data Structure Analysis 131
4.3.7.3.2.2 Data Flow Analysis 131
4.3.7.3.2.3 Control Flow Analysis 133
4.3.7.3.2.4 Interface Analysis 133
4.3.7.3.2.5 Interrupt Analysis 134
4.3.7.3.2.6 Analysis by Inspection 135
4.3.7.3.2.7 Code Analysis Software Tools 138
4.3.7.3.2.8 Formal Proofs of Correctness 139
4.3.8 System Hazard Analysis 139
4.4 Software Safety Testing and Risk Assessment 145
4.4.1 Software Safety Test Planning 146
4.4.1.1 Guidance for Safety-Critical Software Testing 147
4.4.1.2 Nominal and Functional Requirements-Based Testing 149
4.4.1.3 Robustness Testing 149
4.4.1.4 Requirements Coverage Analysis 150
4.4.1.5 Structural Coverage Analysis 151
4.4.1.6 Formal Safety Qualification Testing 151
4.4.2 Software Systems Safety Tests 153

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.4.2.1 Requirements-Based Tests ... 153
4.4.2.2 Functionality Tests ... 154
4.4.2.3 Path Coverage Testing ... 154
4.4.2.4 Statement Coverage Testing ... 155
4.4.2.5 Stress Testing ... 155
4.4.2.6 Endurance Testing ... 155
4.4.2.7 User Interface Tests ... 156
4.4.2.8 Fault Insertion and Failure Mode Tests ... 157
4.4.2.9 Boundary Condition Tests ... 158
4.4.2.10 GO/NO-GO Path Tests ... 160
4.4.2.11 Mutation Testing ... 161
4.4.2.12 Perturbation Testing ... 162
4.4.2.13 Test Phases ... 162
4.4.2.14 Regression Testing ... 164
4.4.3 Software Standards and Criteria Assessment ... 165
4.4.4 Safety Residual Risk Assessment ... 167
4.5 Safety Assessment Report ... 170
4.5.1 SAR Table of Contents ... 171
APPENDIX A DEFINITION OF TERMS ... A-1
A.1 Acronyms ... A-1
A.2 Definitions ... A-6
APPENDIX B REFERENCES ... B-1
B.1 Government References ... B-1
B.2 Commercial References ... B-2
B.3 Individual References ... B-3
B.4 Other References ... B-5
APPENDIX C HANDBOOK SUPPLEMENTAL INFORMATION ... C-1
C.1 DoD Authority and Standards ... C-1
C.1.1 Department of Defense Directive 5000.01 ... C-1
C.1.2 Department of Defense Instruction 5000.02 ... C-1
C.1.3 Defense Acquisition Guidebook ... C-3
C.1.4 Military Standards ... C-4
C.1.4.1 MIL-STD-882B, Notice 1 ... C-4
C.1.4.2 MIL-STD-882C ... C-5
C.1.4.3 MIL-STD-882D ... C-6
C.1.4.4 DOD-STD-2167A ... C-7
C.1.4.5 MIL-STD-498 ... C-7
C.1.5 Other Government Agencies ... C-8
C.1.5.1 Department of Transportation ... C-8
C.1.5.1.1 Federal Aviation Administration ... C-8
C.1.5.1.2 Aerospace Recommended Practice ... C-9
C.1.5.2 Department of Homeland Security ... C-11
C.1.5.2.1 Coast Guard ... C-11
C.1.5.3 National Aeronautics and Space Administration ... C-11
C.1.5.4 Food and Drug Administration ... C-12
C.1.6 Commercial ... C-12

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

C.1.6.1 Institute of Electrical and Electronic Engineering C-12
C.1.6.1.1 IEEE STD 1228-1994 ..... C-12
C.1.6.1.2 IEEE/EIA STD 12207 ..... C-13
C.1.6.2 TechAmerica G-48 System Safety Committee C-13
C.1.6.3 International Electrotechnical Commission C-14
C.2 International Standards C-14
C.2.1 Australian Defense Standard 5679 ..... C-14
C.2.2 United Kingdom Defence Standard 00-56 ..... C-15
C.3 Proposed Contents of the System Safety Data Library C-16
C.3.1 System Safety Program Plan C-16
C.3.2 Software Safety Program Plan C-17
C.3.3 Preliminary Hazard List C-19
C.3.4 Safety-Critical Functions List C-20
C.3.5 Preliminary Hazard Analysis C-21
C.3.6 Subsystem Hazard Analysis C-22
C.3.7 System Hazard Analysis C-23
C.3.8 Safety Requirements Verification Report C-24
C.4 Contractual Documentation C-25
C.4.1 Statement of Operational Need C-25
C.4.2 Request for Proposal C-26
C.4.3 Contract C-26
C.4.4 Statement of Work C-27
C.4.5 System and Product Specification C-28
C.4.6 System and Subsystem Requirements C-29
C.5 Planning Interfaces C-30
C.5.1 Engineering Management C-30
C.5.2 Design Engineering C-30
C.5.3 Systems Engineering C-31
C.5.4 Software Development C-32
C.5.5 Integrated Logistics Support C-32
C.5.6 Other Engineering Support C-33
C.6 Meetings and Reviews C-33
C.6.1 Program Management Reviews C-33
C.6.2 Integrated Product Team Meetings C-34
C.6.3 System Requirements Reviews C-34
C.6.4 System and Subsystem Design Reviews C-35
C.6.5 Preliminary Design Review C-35
C.6.6 Critical Design Review C-36
C.6.7 Test Readiness Review C-37
C.6.8 Functional Configuration Audit C-38
C.6.9 Physical Configuration Audit C-38
C.7 Working Groups C-39
C.7.1 System Safety Working Group C-39
C.7.2 Software System Safety Working Group C-40
C.7.3 Test Integration Working Group/Test Planning Working Group C-41
C.7.4 Computer Resources Working Group C-41

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

C.7.5 Interface Control Working Group C-42
C.8 Resource Allocation C-42
C.8.1 Safety Personnel C-42
C.8.2 Funding C-43
C.8.3 Safety Schedules and Milestones C-43
C.8.4 Safety Tools and Training C-44
C.8.5 Required Hardware and Software C-45
C.9 Program Plans C-45
C.9.1 Risk Management Plan C-46
C.9.2 Quality Assurance Plan or Equivalent C-46
C.9.3 Reliability Engineering Plan C-47
C.9.4 Software Development Plan C-48
C.9.5 Systems Engineering Management Plan C-48
C.9.6 Test and Evaluation Master Plan C-50
C.9.7 Software Test Plan C-51
C.9.8 Software Installation Plan C-51
C.9.9 Software Transition Plan C-52
C.10 Hardware and Human Interface Requirements C-52
C.10.1 Interface Requirements C-52
C.10.2 Operations and Support Requirements C-53
C.10.3 Safety and Warning Device Requirements C-53
C.10.4 Protective Equipment Requirements C-54
C.10.5 Procedures and Training Requirements C-54
C.11 Managing Change C-54
C.11.1 Software Defect Resolution C-55
C.11.2 Technology Insertion and Refresh C-56
C.11.3 Software Configuration Control Board C-58
APPENDIX D COTS AND NON-DEVELOPMENTAL ITEM SOFTWARE D-1
D.1 Introduction D-1
D.2 Characteristics of COTS Software D-1
D.2.1 Advantages of COTS Software D-2
D.2.2 Disadvantages of COTS Software D-2
D.3 Making a COTS Use Decision D-3
D.3.1 Confidence D-4
D.3.2 Influence D-5
D.3.3 Complexity D-7
D.4 COTS Software Safety Selection Process Example D-8
D.4.1 Confidence Metric D-8
D.4.2 Influence Metric D-9
D.4.3 Complexity Metric D-10
D.4.4 Measure of Success D-13
D.5 COTS/NDI Software Safety Process Implementation D-13
D.6 Safety Risk Reduction Requirements D-13
D.6.1 Applications Software Design D-14
D.6.2 Middleware or Wrappers D-15
D.6.3 Message Protocol D-16

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

D.6.4 Designing Around COTS D-16
D.6.5 Analysis and Test of COTS Software D-17
D.6.6 Eliminating Functionality D-17
D.6.7 Run-Time Versions D-17
D.6.8 Watchdog Timers D-18
D.6.9 Configuration Management D-18
D.6.10 Prototyping D-19
D.6.11 Testing D-19
D.7 Safety Case Development for COTS/NDI Software D-19
D.8 Summary D-20
APPENDIX E GENERIC SOFTWARE SAFETY REQUIREMENTS AND GUIDELINES E-1
E.1 Introduction E-1
E.1.1 Determination of Safety-Critical Computing System Functions E-1
E.1.1.1 Specifications E-1
E.1.2 Safety Criticality E-1
E.1.3 Identification of Safety-Critical Computing System Functions E-2
E.1.3.1 Safety Application Functions E-2
E.1.3.2 Safety Infrastructure Functions E-3
E.1.3.3 Common SCCSFs E-3
E.1.4 Requirement Types E-4
E.1.4.1 Behavioral E-4
E.1.4.2 Non-Behavioral E-5
E.1.4.3 Design Constraint E-5
E.1.5 Implementation of Generic Requirements and Guidelines E-6
E.1.5.1 Integration with the Software Engineering Process E-6
E.1.5.2 Audit Tool Post-Software Design or Development E-6
E.1.5.3 Full Compliance vs. Partial or Non-Compliance E-6
E.2 Design and Development Process Requirements and Guidelines E-7
E.2.1 Configuration Control E-7
E.2.2 Software Quality Assurance Program E-7
E.2.3 Two Person Rule E-7
E.2.4 Program Patch and Overlay Prohibition E-8
E.2.5 Software Design Verification and Validation E-8
E.2.5.1 Correlation of Artifacts Analyzed to Artifacts Deployed E-8
E.2.5.2 Correlation of Process Reviewed to Process Employed E-9
E.2.5.3 Reviews and Audits E-10
E.3 System Design Requirements and Guidelines E-10
E.3.1 Designed Safe States E-10
E.3.2 Safe State Return E-10
E.3.3 Stand-Alone Computer E-10
E.3.4 Ease of Maintenance E-11
E.3.5 Restoration of Interlocks E-11
E.3.6 Input and Output Registers E-11
E.3.7 External Hardware Failures E-11
E.3.8 Safety Kernel Failure E-11

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.3.9 Circumvent Unsafe Conditions E-11
E.3.10 Fallback and Recovery E-12
E.3.11 Simulators E-12
E.3.12 System Errors Log E-12
E.3.13 Positive Feedback Mechanisms E-12
E.3.14 Peak Load Conditions E-12
E.3.15 Endurance Issues E-12
E.3.16 Error Handling E-13
E.3.17 Redundancy Management E-15
E.3.18 Safe Modes and Recovery E-15
E.3.19 Isolation and Modularity E-16
E.4 Power-Up System Initialization Requirements E-16
E.4.1 Power-Up Initialization E-17
E.4.2 Power Faults E-17
E.4.3 Primary Computer Failure E-17
E.4.4 Maintenance Interlocks E-17
E.4.5 System-Level Check E-17
E.4.6 Control Flow Defects E-17
E.5 Computing System Environment Requirements and Guidelines E-20
E.5.1 Hardware and Hardware/Software Interface Requirements E-20
E.5.1.1 Failure in the Computing Environment E-20
E.5.2 CPU Selection E-21
E.5.3 Minimum Clock Cycles E-21
E.5.4 Read Only Memory E-22
E.6 Self-Check Design Requirements and Guidelines E-22
E.6.1 Watchdog Timers E-22
E.6.2 Memory Checks E-22
E.6.3 Fault Detection E-22
E.6.4 Operational Checks E-22
E.7 Safety-Critical Computing System Functions Protection Requirements and Guidelines E-23
E.7.1 Safety Degradation E-23
E.7.2 Unauthorized Interaction E-23
E.7.3 Unauthorized Access E-23
E.7.4 Safety Kernel ROM E-23
E.7.5 Safety Kernel Independence E-23
E.7.6 Inadvertent Jumps E-23
E.7.7 Load Data Integrity E-24
E.7.8 Operational Reconfiguration Integrity E-24
E.8 Interface Design Requirements E-24
E.8.1 Feedback Loops E-24
E.8.2 Interface Control E-24
E.8.3 Decision Statements E-24
E.8.4 Inter-CPU Communications E-24
E.8.5 Data Transfer Messages E-25
E.8.6 External Functions E-25

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.8.7 Input Reasonableness Checks E-25
E.8.8 Full-Scale Representations....E-25
E.9 Human Interface....E-25
E.9.1 Operator/Computing System Interface E-25
E.9.1.1 CHI Issues E-26
E.9.2 Processing Cancellation E-27
E.9.3 Hazardous Function Initiation....E-27
E.9.4 Safety-Critical Displays E-27
E.9.5 Operator Entry Errors E-27
E.9.6 Safety-Critical Alerts E-27
E.9.7 Unsafe Situation Alerts E-28
E.9.8 Unsafe State Alerts E-28
E.10 Critical Timing and Interrupt Functions E-28
E.10.1 Safety-Critical Timing E-28
E.10.2 Valid Interrupts E-28
E.10.3 Recursive Loops....E-28
E.10.4 Time Dependency E-29
E.11 Software Design and Development Requirements and Guidelines E-29
E.11.1 Coding Requirements and Issues E-29
E.11.1.1 Ada Language Issues E-30
E.11.2 Modular Code E-31
E.11.3 Number of Modules E-31
E.11.4 Execution Path E-31
E.11.5 Halt Instructions....E-32
E.11.6 Single Purpose Files....E-32
E.11.7 Unnecessary Features....E-32
E.11.8 Indirect Addressing Methods....E-32
E.11.9 Uninterruptible Code E-32
E.11.10 Safety-Critical Files E-32
E.11.11 Unused Memory....E-32
E.11.12 Overlays of Safety-Critical Software Shall Occupy the Same Amount of Memory....E-33
E.11.13 Operating System Functions E-33
E.11.14 Compilers....E-33
E.11.15 Flags and Variables....E-33
E.11.16 Loop Entry Point....E-33
E.11.17 Software Maintenance Design E-33
E.11.18 Variable Declaration E-34
E.11.19 Unused Executable Code E-34
E.11.20 Unreferenced Variables E-34
E.11.21 Assignment Statements....E-34
E.11.22 Conditional Statements E-34
E.11.23 Strong Data Typing....E-34
E.11.24 Timer Values Annotated....E-35
E.11.25 Critical Variable Identification E-35
E.11.26 Global Variables E-35

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.12 Software Maintenance Requirements and Guidelines E-35
E.12.1 Critical Function Changes....E-35
E.12.2 Critical Firmware Changes E-35
E.12.3 Software Change Medium E-36
E.12.4 Modification Configuration Control E-36
E.12.5 Version Identification E-36
E.13 Software Analysis and Testing E-36
E.13.1 General Testing Guidelines....E-36
E.13.2 Trajectory Testing for Embedded Systems....E-37
E.13.3 Formal Test Coverage....E-38
E.13.4 Go/No-Go Path Testing E-38
E.13.5 Input Failure Modes....E-38
E.13.6 Boundary Test Conditions E-38
E.13.7 Input Rata Rates....E-38
E.13.8 Zero Value Testing E-39
E.13.9 Regression Testing....E-39
E.13.10 Operator Interface Testing E-39
E.13.11 Duration Stress Testing....E-39
APPENDIX F LESSONS LEARNED....F-1
F.1 Therac Radiation Therapy Machine Fatalities....F-1
F.1.1 Summary....F-1
F.1.2 Key Facts F-1
F.1.3 Lessons Learned....F-2
F.2 Missile Launch Timing Error Causes Hang-Fire....F-2
F.2.1 Summary....F-2
F.2.2 Key Facts F-3
F.2.3 Lessons Learned....F-3
F.3 Reused Software Causes Flight Controls to Shut Down F-3
F.3.1 Summary....F-3
F.3.2 Key Facts F-4
F.3.3 Lessons Learned....F-4
F.4 Flight Controls Fail at Supersonic Transition....F-4
F.4.1 Summary....F-4
F.4.2 Key Facts F-5
F.4.3 Lessons Learned....F-5
F.5 Incorrect Missile Firing from Invalid Setup Sequence F-6
F.5.1 Summary....F-6
F.5.2 Key Facts F-6
F.5.3 Lessons Learned....F-6
F.6 Operator's Choice of Weapon Release Overridden by Software Control F-7
F.6.1 Summary....F-7
F.6.2 Key Facts F-7
F.6.3 Lessons Learned....F-8
APPENDIX G EXAMPLE REQUEST FOR PROPOSAL AND STATEMENT OF WORK G-1
G.1 Sample RFP G-1

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

G.2 Sample Statement of Work G-2
G.2.1 System Safety G-2
G.2.2 Software Safety G-3

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table of Figures

Figure 2-1: Management Commitment to the Integrated Safety Process 7
Figure 2-2: Example of Internal System Interfaces 8
Figure 2-3: Defense Acquisition Management Framework..... 9
Figure 2-4: Relationship of Software to the Hardware Development Lifecycle 10
Figure 2-5: Grand Design Waterfall Software Acquisition Lifecycle Model..... 12
Figure 2-6: Modified V Software Acquisition Lifecycle Model 13
Figure 2-7: Spiral Software Acquisition Lifecycle Model 15
Figure 2-8: Integration of Engineering Personnel and Processes 19
Figure 2-9: Handbook Layout..... 24
Figure 2-10: Section 4 Format 25
Figure 3-1: Types of Risk 29
Figure 3-2: Systems Engineering and Risk Management Documentation 32
Figure 3-3: Mishap Risk Example 40
Figure 3-4: Hazard Reduction Order of Precedence..... 45
Figure 4-1: Chapter 4 Contents..... 48
Figure 4-2: Software System Safety Interfaces 49
Figure 4-3: Process Chart Format Example..... 51
Figure 4-4: Software Safety Planning..... 53
Figure 4-5: Software Safety Planning by the PA..... 55
Figure 4-6: Software Safety Planning by the DA 57
Figure 4-7: Planning the Safety Criteria is Important..... 59
Figure 4-8: Two Distinct Processes of Software Safety Engineering..... 61
Figure 4-9: Graphical Depiction of Software Assurance and Integrity 62
Figure 4-10: Graphical Depiction of Software Safety Hazard Analysis..... 64
Figure 4-11: Legacy Software Control Category Definitions..... 65
Figure 4-12: Recommended Terms and Definitions..... 68
Figure 4-13: Example LOR Template 70
Figure 4-14: Software Safety Program Interfaces 74
Figure 4-15: Proposed SSS Team Membership..... 77
Figure 4-16: Example of Risk Acceptance Matrix 80
Figure 4-17: Likelihood of Occurrence Example 82
Figure 4-18: Software Safety Program Management 84
Figure 4-19: Software Safety Task Implementation..... 88
Figure 4-20: Example POA&M Schedule 90
Figure 4-21: Functional Hazard Analysis 94
Figure 4-22: An Example of Safety-Critical Functions..... 95
Figure 4-23: PHL Development..... 97
Figure 4-24: PHA..... 99
Figure 4-25: Hazard Analysis Segment 101
Figure 4-26: Example of a PHA Format..... 103
Figure 4-27: Safety Requirements Analysis 104
Figure 4-28: SSR Derivation 105
Figure 4-29: Tailoring the Generic Safety Requirements..... 108

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Figure 4-30: Example Software Safety Requirements Tracking Worksheet 109
Figure 4-31: In-Depth Hazard Causal Analysis 112
Figure 4-32: Preliminary Software Design Analysis 114
Figure 4-33: SSR Verification Tree 116
Figure 4-34: Hierarchy Tree Example 121
Figure 4-35: Detailed Software Design Analysis 123
Figure 4-36: Verification Methods 125
Figure 4-37: Identification of Safety-Significant CSUs 126
Figure 4-38: Example of a DFD 132
Figure 4-39: Flow Chart Examples 133
Figure 4-40: SHA 141
Figure 4-41: Example of a SHA Interface Analysis 142
Figure 4-42: Documentation of Interface Causes and Safety Requirements 144
Figure 4-43: Software Safety Test Planning 146
Figure 4-44: Software Safety Testing and Analysis 148
Figure 4-45: Software Regression Testing 165
Figure 4-46: Software Requirements Verification 166
Figure 4-47: Software's Contribution to Residual Safety Risk Assessment 169
Figure C-1: Contents of an SwSPP - IEEE STD 1228-1994 ..... C-19
Figure C-2: SSHA and SHA Hazard Record Example C-23
Figure C-3: Hazard Requirements Verification Document Example C-25
Figure C-4: Software Safety SOW Paragraphs C-28
Figure C-5: Software Defect Resolution C-56
Figure C-6: Technology Insertion and Refresh C-57
Figure C-7: Generic Software Configuration Change Process C-59
Figure D-1: Initial Considerations for COTS Selection D-3
Figure D-2: COTS Evaluation Space D-4
Figure D-3: Example COTS Confidence Criteria Metric D-9
Figure D-4: Example Safety Influence Metric D-10
Figure D-5: Example Safety Factors of Complexity D-11
Figure D-6: Example Testability Factors of Complexity D-12
Figure D-7: Example Integration Factors of Complexity D-12

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table of Tables

Table 3-1: Severity Categories 41
Table 3-2: Probability Levels 42
Table 3-3: Risk Assessment Matrix 43
Table 4-1: Software Control Category Definitions 66
Table 4-2: Software Criticality Matrix 69
Table 4-3: Example LOR Tasks or Requirements 71
Table 4-4: Example of Specific LOR Tasks 73
Table 4-5: Example FHA Template 96
Table 4-6: Acquisition Process Trade-Off Analyses 100
Table 4-7: Example of an SSR Verification Matrix 117
Table 4-8: Example of an RTM 117
Table 4-9: Example Safety-Significant Function Matrix 120
Table 4-10: Data Item Example 131

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The Office of Primary Responsibility for this guide is the Office of the Under Secretary of Defense for Acquisition, Technology and Logistics. Special thanks to Mr. Chris DiPetto, Dr. Liz Rodriguez-Johnson, and the Director of Defense Research and Engineering for their roles in the development of this Handbook.

DoD Office of Primary Responsibility for completion of Handbook
Archibald McKinlay, VI

1999 Original Authors
David Alberico
John Bozarth
Michael Brown
Janet Gill
Steven Mattern

2007 Returning Original Authors
Michael Brown
Dr. Janet Gill
Steven Mattern
Archibald McKinlay, VI

2007-2010 New Contributing Authors and Technical Editors

Industry
Richard Church
Murray Donaldson
Rene Fitzpatrick
Christine Hines
Samuel Redwine
Paige Ripani
Frank Salvatore
Robert Schmedake
Sherry Smith
Kristin Thompson
Kathryn G. Whorf

U.S. Army
Jeffrey Marc Fornoff
Patty Lyon
David Magidson
Frank Marotta
Josh McNeil
Gohan Oraby
Sandy Picinic
Dr. George Vinansky

U.S. Coast Guard
Chris Kijora

NASA
Steven D. Smith
Daniel C. Victor
Martha Wetherholt
John C. Wolf

U.S. Navy
Mary Ellen Caro
Karen Cooper
Robert Heflin
Archibald McKinlay, VI
David Shampine

U.S. Marine Corps
Scott Rideout

Missile Defense Agency
Grady Lee
Steven Pereira

U.S. Air Force
Robert Baker
Todd Griffin
Douglas Peterson
Daniel Strub
Jeffrey Wethern

2010 Core Team
Karen Cooper
Michelle DePrenger
Steven Mattern
Archibald McKinlay, VI
Azi Pajouhesh
David Shampine

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

1 Overview

Since the development of the digital computer and computing devices, software and firmware logic continues to play an important and evolutionary role in the operation and control of hazardous, safety-critical functions (SCFs). The reluctance of the engineering community to relinquish human control of hazardous operations has diminished over the last 25 years. Today, digital computer systems have autonomous control over safety-critical functions in nearly every major technology, both commercially and within Government systems. This revolution is due primarily to the ability of software to perform critical control tasks reliably at speeds unmatched by its human counterpart. Other factors influencing this transition are the ever-growing need for increased versatility, higher performance capability, greater efficiency, increased network interoperability, and decreased lifecycle cost. In most instances, properly designed software can meet all of these attributes for system performance. The logic of the software allows for decisions to be implemented with speed and accuracy without the human operator in the decision-making loop.

Within the domain of systems engineering, systems safety engineering identifies and analyzes behavioral and interface requirements, the design architecture, and the human interface within the context of both systems and systems of systems (SoS). In addition, system safety engineering defines requirements for design and systems engineering, taking into account the potential risks, verification and validation (V&V) of effective mitigation, and residual risk acceptance by certification or approval authorities.

It is essential to perform system safety engineering tasks on safety-critical systems to reduce safety risk in all aspects of a program. These tasks include software system safety activities involving the design, code, test, independent verification and validation (IV&V), operation and maintenance, and change control functions within the software engineering development and deployment processes.

The main objective of system safety engineering, which includes software system safety, is the application of engineering and management principles, criteria, and techniques to optimize all aspects of safety within the constraints of operational effectiveness, time, and cost throughout all phases of the system lifecycle.

Program management is ultimately responsible for the development of a safe system. The commitment to provide qualified personnel and an adequate budget and schedule for a software development program begins with the program director or Program Manager (PM). Senior management must be a strong voice of safety advocacy and must communicate this commitment to each level of program and technical management. The PM must provide the necessary resources to support the integrated safety process between systems engineering, software engineering, and safety engineering in the design, development, test, operation, and maintenance of the system software. The purpose of this Joint Software Systems Safety Engineering Handbook (JSSSEH), hereafter referred to as the Handbook, is to provide management and

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

engineering guidelines to achieve a reasonable level of assurance that software will execute within the system context with an acceptable level of safety risk.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

2 Introduction to the Handbook

2.1 Introduction

All members of the system development team should read Section 2 of the Handbook, which discusses the following major subjects:

- The purpose of this Handbook
- The scope of the subject matter this Handbook presents
- The authority by which a Software System Safety (SSS) program is conducted
- How this Handbook is organized for maximum benefit.

As a member of the software development team, the safety engineer is a critical member in the design, redesign, integration, maintenance, and sustainment of modern high consequence systems. Whether the safety engineer is a hardware engineer, software engineer, safety specialist, or safety manager, it is their responsibility to ensure that an acceptable level of safety risk is achieved and maintained throughout the lifecycle of the system(s) being developed. This Handbook provides a rigorous and pragmatic application of SSS planning and analysis for the safety engineer.

SSS, an element of the total system safety and software development programs, cannot function independent of the total effort, nor can it be ignored. Systems, both “simple” and highly integrated with multiple subsystems and SoS, are experiencing an extraordinary growth with the use of computers and software to monitor and control safety-significant subsystems and functions. A software specification error, design flaw, or lack of initial safety design requirements can contribute to or cause a system failure or erroneous human decision. Death, injury, loss of the system or other assets, or environmental damage can result. To achieve an acceptable level of safety for software used in critical applications, software safety engineering must be emphasized early in the requirements definition and system conceptual design process. Safety-significant software must then receive continuous emphasis from management and a continuing integrated engineering analysis and testing process throughout the development and operational lifecycles of the system.

This Handbook is a product of a joint effort. The U.S. Army, Department of the Navy, Air Force, and Coast Guard Safety Centers, with cooperation from the Federal Aviation Administration (FAA), National Aeronautics and Space Administration (NASA), defense industry contractors, and academia, are the primary contributors. This Handbook captures the “best practices” pertaining to SSS program management and safety-critical software design. The Handbook consolidates these best practices into a single and complete resource. The Handbook aids the system development team in understanding its software system safety responsibilities. By using this Handbook, the user will appreciate the need for all disciplines to work together in

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

identifying, controlling, and managing software-significant hazards within the components of hardware systems.

To summarize, this Handbook is an instructional guide for understanding SSS and the contribution of each functional discipline to the overall goal. The Handbook is applicable to all types of systems (military and commercial) in all types of operational uses.

2.2 Purpose

The purpose of the Handbook is to provide management and engineering guidelines to achieve a reasonable level of assurance that the software will execute within the system context with an acceptable level of safety risk.

2.3 Scope

This Handbook is both a reference document and management tool for aiding managers and engineers at all levels in any Government or industrial organization. This Handbook describes how to develop and implement an effective SSS process. This process minimizes the likelihood or severity of system hazards caused by poorly specified, designed, developed, or operated software in safety-significant applications. Furthermore, technology refresh, operational upgrades, and operational risk during sustainment must be controlled or mitigated and are an integral part of the complete SSS process.

The primary responsibility for management of the SSS process lies with the system safety manager or engineer in both the developer's (supplier) and acquirer's (customer) organizations. However, every functional discipline has a vital role and must be involved in the SSS process. The SSS tasks, techniques, and processes outlined in this Handbook can be applied to any system that uses software in critical areas. The JSSSEH highlights the need for all contributing disciplines to understand and apply qualitative analysis techniques to ensure the safety of hardware systems controlled by software.

This Handbook, while extensive, is a guide and is not intended to supersede any Agency policy, standard, or guidance pertaining to system safety (e.g., Military Standard (MIL-STD)-882D) or software engineering and development (e.g., International Organization for Standardization (ISO) 12207). This Handbook is written to clarify the SSS requirements and tasks specified in Government and commercial standards and guidance documents. The Handbook provides the system safety manager and the software development manager with sufficient information to:

- Properly scope the SSS effort in the Statement of Work (SOW)
- Properly integrate the defined SSS tasks into the program's engineering and managements processes for each phase of the acquisition lifecycle

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Identify the data needed to monitor contractor compliance effectively with the contract system safety requirements
- Evaluate contractor performance throughout the development lifecycle.

This Handbook is not a tutorial for software engineering. However, the Handbook addresses some technical aspects of software design and function to assist with understanding software safety. This Handbook will provide each member of the SSS team with a basic understanding of sound systems and software safety practices, processes, and techniques. The JSSSEH will demonstrate the importance of each technical and managerial discipline working together to define software safety requirements (SSR) for the safety-significant software components of the system. The Handbook will also illustrate opportunities where the team can design additional safety features into the software to eliminate or control identified hazards.

2.4 Authority and Standards

Numerous directives, standards, regulations, and regulatory guides establish the authority for system safety engineering requirements in the acquisition, development, and maintenance of software-based systems. Although the primary focus of this Handbook is military systems, much of the authority for the establishment of Department of Defense (DoD) system safety and software safety programs derives from other Governmental and commercial standards and guidance. This Handbook documents and consolidates many of these authoritative standards and guidelines and demonstrates the importance Government places on the reduction of safety risk for software performing safety-significant functions. This allows a PM, safety manager, or safety engineer to clearly demonstrate mandated requirements and the need for a software safety program to their superiors.

Within DoD and the acquisition corps of each Military Service, the primary documents pertaining to system safety and software development include Department of Defense Directive (DoDD) 5000.01, *The Defense Acquisition System*, and MIL-STD-882D, *Standard Practice for System Safety*. Department of Defense Instruction (DoDI) 5000.02, *Operations of the Defense Acquisition System and the Defense Acquisition Guidebook*, is also applicable. Whether a program or project uses MIL-STD-498, *Software Development*, or more currently, ISO 12207, this guidance requires tailoring and adaptation. Applicable guidance and specific instruction from each of these documents are provided in Appendix C. Be advised that the language within these documents changes on a periodic basis as the Government moves to streamline their approach to acquisition and procurement philosophies.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

2.5 Handbook Overview

2.5.1 Historical Background

The introduction of software-controlled, safety-critical systems has caused considerable ramifications in the managerial, technical, safety, economic, and scheduling risks of both hardware and software system development. Software is generally cheaper to develop and maintain than hardware; however, this may not be true if the requirements change or the software is poorly developed. Software is faster, especially with modern processors, than equivalent analog or discrete digital systems and is easier to modify. Computers and software may provide information contrary to warfighter sensory observations in the field. Conflicting information can result in erroneous decision making and increased risk. This makes the potential introduction of safety-significant errors in software all the more critical in modern systems.

Section 3 of this Handbook further discusses risk. Section 4 includes the identification, documentation (e.g., evidence through analyses), and elimination or control (to an acceptable level) of the safety risk associated with software in the design, requirements, development, test, operation, and support of the system.

A software design flaw or run-time error within safety-significant functions of a system introduces the potential for a hazardous condition that could result in death, personal injury, loss of the system, or environmental damage. Appendix F provides abstracts with examples of software-influenced accidents and failures. The incident examples in Appendix F include:

- F.1 - Therac Radiation Therapy Machine Fatalities
- F.2 - Missile Launch Timing Error Causes Hang-Fire
- F.3 - Reused Software Causes Flight Controls to Shut Down
- F.4 - Flight Controls Fail at Supersonic Transition
- F.5 - Incorrect Missile Firing from Invalid Setup Sequence
- F.6 – Operator’s Choice of Weapon Release Overridden by Software Control.

The examples in Appendix F are a small representation of possible software control issues.

The software system safety techniques, methods, and processes continues to mature. This Handbook update identifies and integrates the advancements made as “best practices” within the system safety, software safety, and software assurance communities to further reduce the potential of software contributing to known hazards or mishaps of a given system.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

2.5.2 Management Responsibilities

Program management is ultimately responsible for the development of a safe system. The commitment of qualified individuals and an adequate budget and schedule for the software development program must begin with the program director or PM. Top management must be a strong advocate for safety and must communicate this personal commitment to each level of program and technical management. The PM must support the integrated safety processes within systems engineering and software engineering in the design, development, test, and operation and maintenance of the system software. Figure 2-1 portrays the managerial element for the integrated team. Historically, one of the root causes of software system safety program (SwSSP) failures, and resultant increased levels of safety residual risks, can be traced to a lack of program management and resources to support the software system safety effort at program outset.

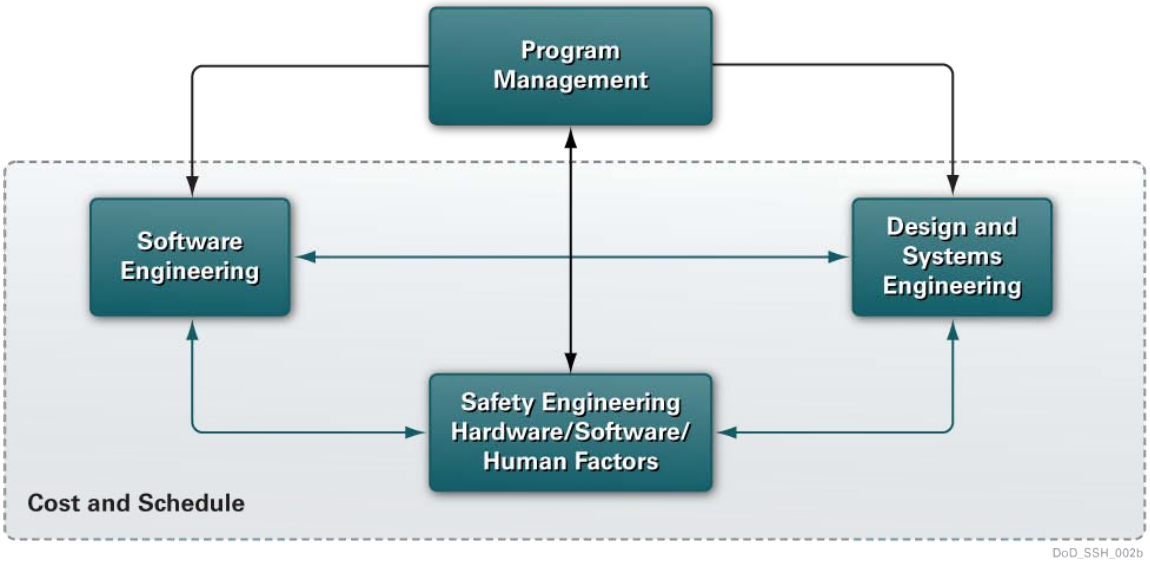

Figure 2-1: Management Commitment to the Integrated Safety Process

2.5.3 Introduction to the Systems Approach

System safety engineering has historically demonstrated the benefits of a “systems” approach to safety risk analysis and mitigation. When conducting a hazard analysis on a hardware subsystem as a separate entity, safety engineering identifies a set of unique hazards applicable only to that subsystem. However, when analyzing that same subsystem in the context of its physical, functional, and zonal interfaces with other system components, the analysis may identify additional hazards or hazard causal factors not noted in the original analysis.

Conversely, the results of a system analysis may demonstrate that hazards identified in the subsystem analysis were either reduced or eliminated by other components of the system. The

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

identification of critical subsystem interfaces (such as software) and their contribution to associated hazards is a vital aspect of safety risk minimization for the total system.

Analyzing software that performs or controls safety-significant functions within a system requires a systems approach. The success of a software safety program is predicated on this approach. Software is a critical component of the safety risk potential of modern systems being developed and fielded. Both the internal and external interfaces of the system are important to safety.

Figure 2-2 depicts specific software internal interfaces within the system block, as well as the external software inputs. Each software interface or input may possess safety risk potential to the operators, maintainers, environment, or the system itself. The acquisition and development process must consider these interfaces during the design of both the hardware and software systems. To accomplish this, the design team must fully understand and integrate the hardware and software development lifecycles.

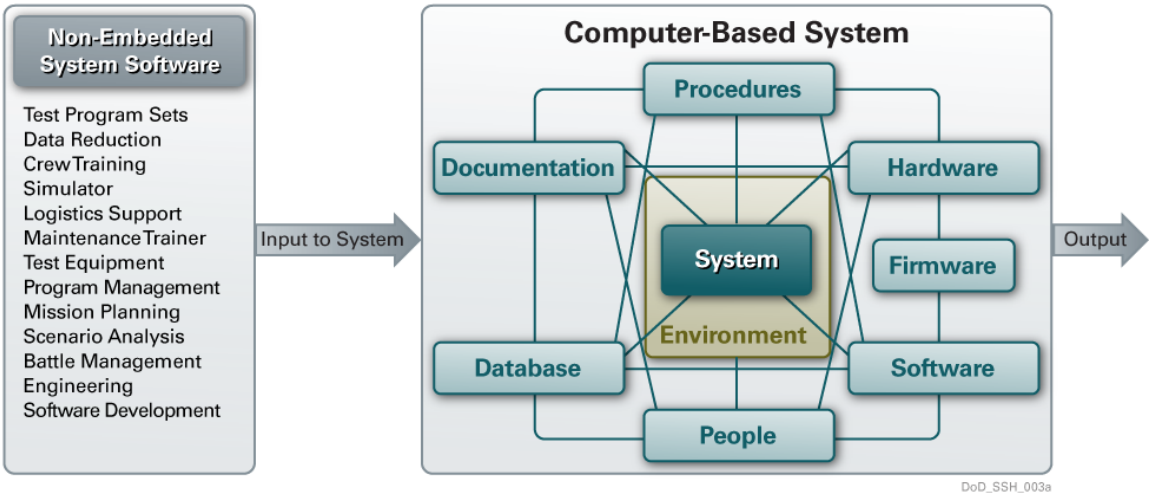

Figure 2-2: Example of Internal System Interfaces

2.5.3.1 The Hardware Development Lifecycle

The typical hardware development lifecycle has been in existence for many years. In most cases, this proven acquisition model produced the desired engineering results in the design, development, manufacturing, fabrication, and test activities. The lifecycle consists of five phases— (1) material solution analysis, (2) technology development, (3) system engineering and manufacturing development, (4) production and deployment, and (5) operations and support. Each phase of the lifecycle ends and the next phase begins with a milestone (MS) decision point (e.g., Concept Decision; Milestones A, B, and C). The Milestone Decision Authority (MDA) makes an assessment of the system design and program status at each milestone decision point, and makes or reviews plans for subsequent phases of the lifecycle. DoD Acquisition Instructions

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

and associated regulations and guidebooks provide requirements and guidance to the system acquisition process.

The purpose of introducing the system lifecycle in this Handbook is to familiarize the reader with a typical lifecycle model. Current DoD procurements use the Defense Acquisition Management Framework shown in Figure 2-3. The framework identifies and establishes defined phases for the development lifecycle of a system and allows development teams to overlay the lifecycle on a proposed timetable to establish the milestone schedule. Defense Systems Management College (DSMC) documentation and the aforementioned acquisition instructions and regulations provide detailed information regarding milestones and phases of a system lifecycle and systems acquisition management course documentation of the individual Services. It is critical that system safety engineering schedules, tasks, and artifacts align with and support the acquisition schedule.

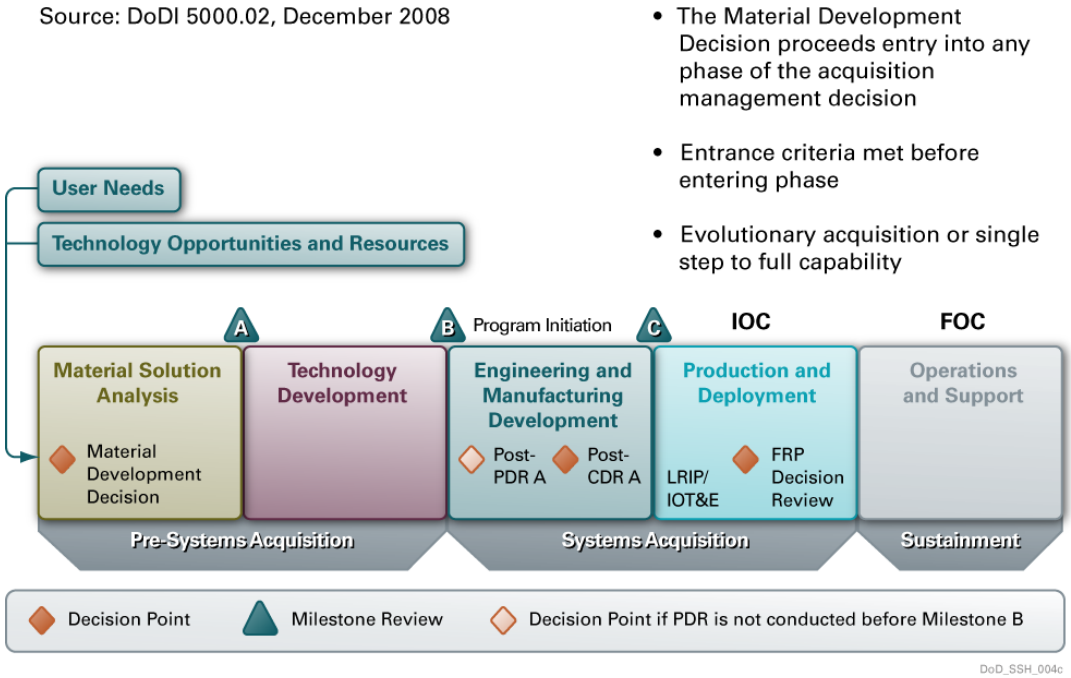

Figure 2-3: Defense Acquisition Management Framework

2.5.3.2 The Software Development Lifecycle

The system safety team must be fully aware of the software lifecycle used by the development activity or team. In the past several years, numerous lifecycle models have been identified, modified, and used on a variety of software development programs. This Handbook will not discuss the merits and limitations of different lifecycle process models because the software engineering team must make the decision for or against a model for an individual procurement. The important point is that the system safety team must recognize the model being used and appropriately correlate and integrate safety activities into the model to achieve the desired

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

outcomes and safety goals. The following paragraphs present several different models and examples of the various models to the reader. However, this is not an exhaustive treatment of the models and variations of models available.

Figure 2-4 is a graphical representation of the relationship of the software development lifecycle to the system or hardware development lifecycle. Note that the software lifecycle graphic shown in Figure 2-4 portrays a typical DoD standard software lifecycle. The model is representative of the "Waterfall" or "Grand Design" lifecycle. While this model is not current, it is still in use on numerous procurements. Other models are more representative of the software development schemes currently being followed, such as the "Spiral" rapid prototyping and "Modified V" software development lifecycles, Object-Oriented Analysis and Design (OOA&D), and other techniques.

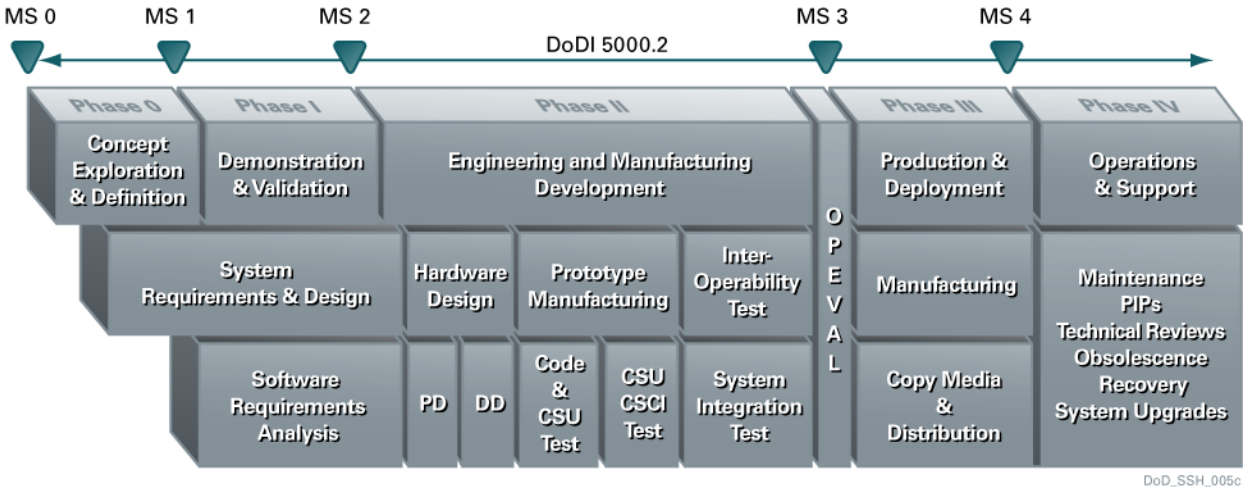

Figure 2-4: Relationship of Software to the Hardware Development Lifecycle

It is important to recognize that the software development lifecycle does not correlate exactly with the hardware (system) development lifecycle. Software development lags the hardware development at the beginning of the process but generally finishes before hardware development is completed. Specific design reviews for hardware may lag those required for software. Section 4 of this Handbook discusses the implications of this disparity.

2.5.3.2.1 Grand Design and Waterfall Lifecycle Model[1]

The Waterfall software acquisition and development lifecycle model is the oldest in terms of use by software developers. This strategy usually is based on terminology used during the early 1970s as a remedy to the ad hoc, *code-and-fix* method of software development. Grand Design

1 Unless otherwise noted, the descriptions of the software acquisition lifecycle models are either quoted or paraphrased from *Guidelines for Successful Acquisition and Management of Software Intensive Systems*; Software Technology Support Center (STSC); September 1994.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

places emphasis on upfront documentation during early development phases, but does not support modern development practices such as prototyping, OOA&D, and automatic code generation. "With each activity as a prerequisite for succeeding activities, this strategy is a risky choice for unprecedented systems because it inhibits flexibility."

Another limitation to the model is that the system is complete after a single pass through the model. Therefore, many integration issues are identified too late in the development process to be corrected without significant cost and schedule impacts. In terms of software safety, interface issues must be identified and rectified as early as possible in the development lifecycle to be adequately corrected and verified.

Figure 2-5 is a representation of the Grand Design, or Waterfall, lifecycle model. The Waterfall model is not recommended for large, software-intensive systems because of the limitations stated above and the inability to effectively manage program risks, including safety risk, during the software development process. However, the Grand Design lifecycle model does provide a structured and well-disciplined method for software development.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

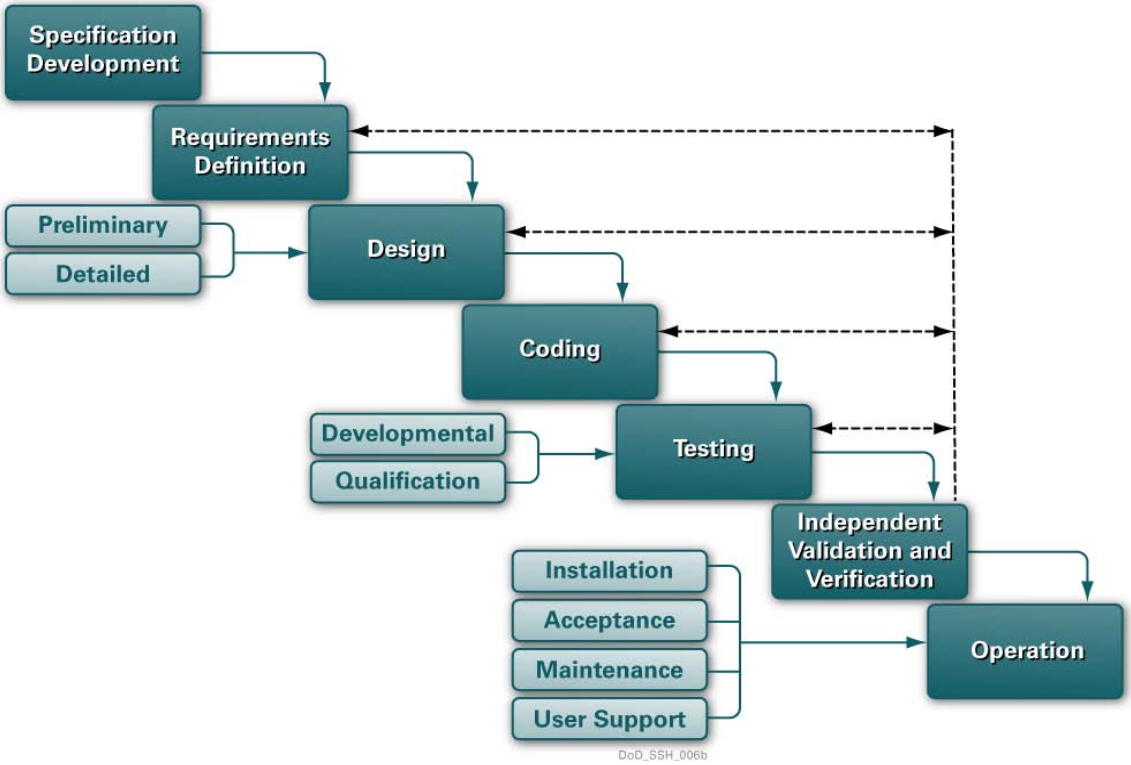

Figure 2-5: Grand Design Waterfall Software Acquisition Lifecycle Model

2.5.3.2.2 Modified V Lifecycle Model[2]

The Modified V software acquisition lifecycle model, shown in Figure 2-6, is another example of a defined method for software development. This model is heavily weighted in the ability to design, code, prototype, and test in increments of design maturity. The left side of the figure identifies the specification, design, and coding activities for developing software. This side also indicates when the test specification and test design activities can begin. For example, the system and acceptance tests can be specified and designed as soon as software requirements are known. The integration tests can be specified and designed as soon as the software design structures are known. The unit tests can be specified and designed when the code units are prepared.[3] The right side of the figure identifies when the evaluation activities occur that are involved with the execution and testing of the code at various stages of evolution.

2 Unless otherwise noted, the descriptions of the software acquisition lifecycle models are either quoted or paraphrased from *Guidelines for Successful Acquisition and Management of Software Intensive Systems*; STSC; September 1994.
3 *Software Test Technologies Report*, STSC; Hill Air Force Base, Utah; August 1994.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

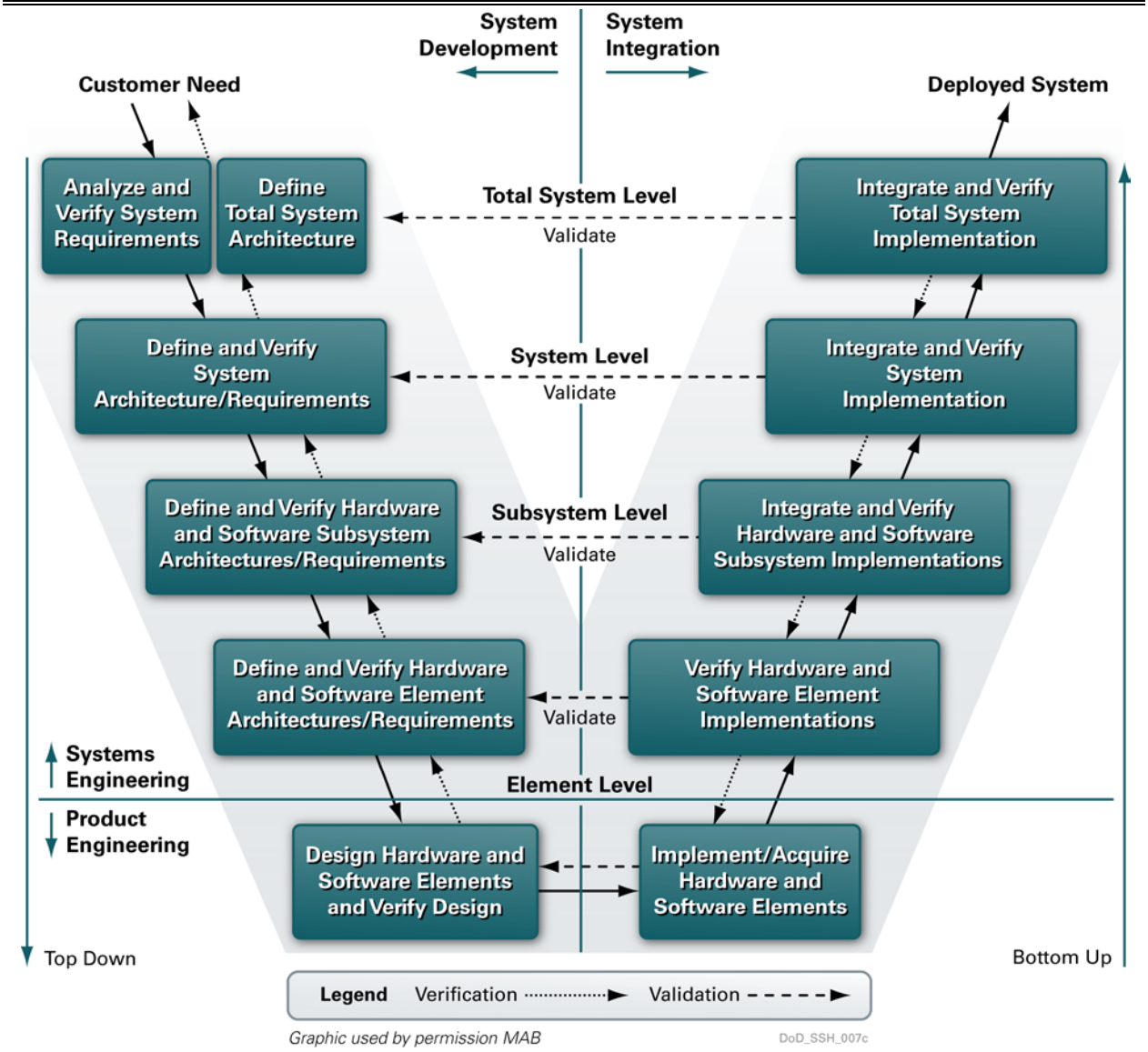

Figure 2-6: Modified V Software Acquisition Lifecycle Model

2.5.3.2.3 Spiral Lifecycle Model

The Spiral acquisition lifecycle model provides a risk-reduction approach to the software development process. In the Spiral model, Figure 2-7, the radial distance is a measure of effort expended, while the angular distance represents progress made. The model combines features of the Waterfall and the incremental prototype approaches to software development. Spiral development emphasizes evaluation of alternatives and risk assessment. These issues are addressed more thoroughly than with other strategies. A review at the end of each phase ensures commitment to the next phase or identifies the need to rework a phase if necessary. The

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

advantages of Spiral development are the emphasis on procedures, such as risk analysis, and the adaptability to different development approaches.[4]

4 *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, STSC; September 1994.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

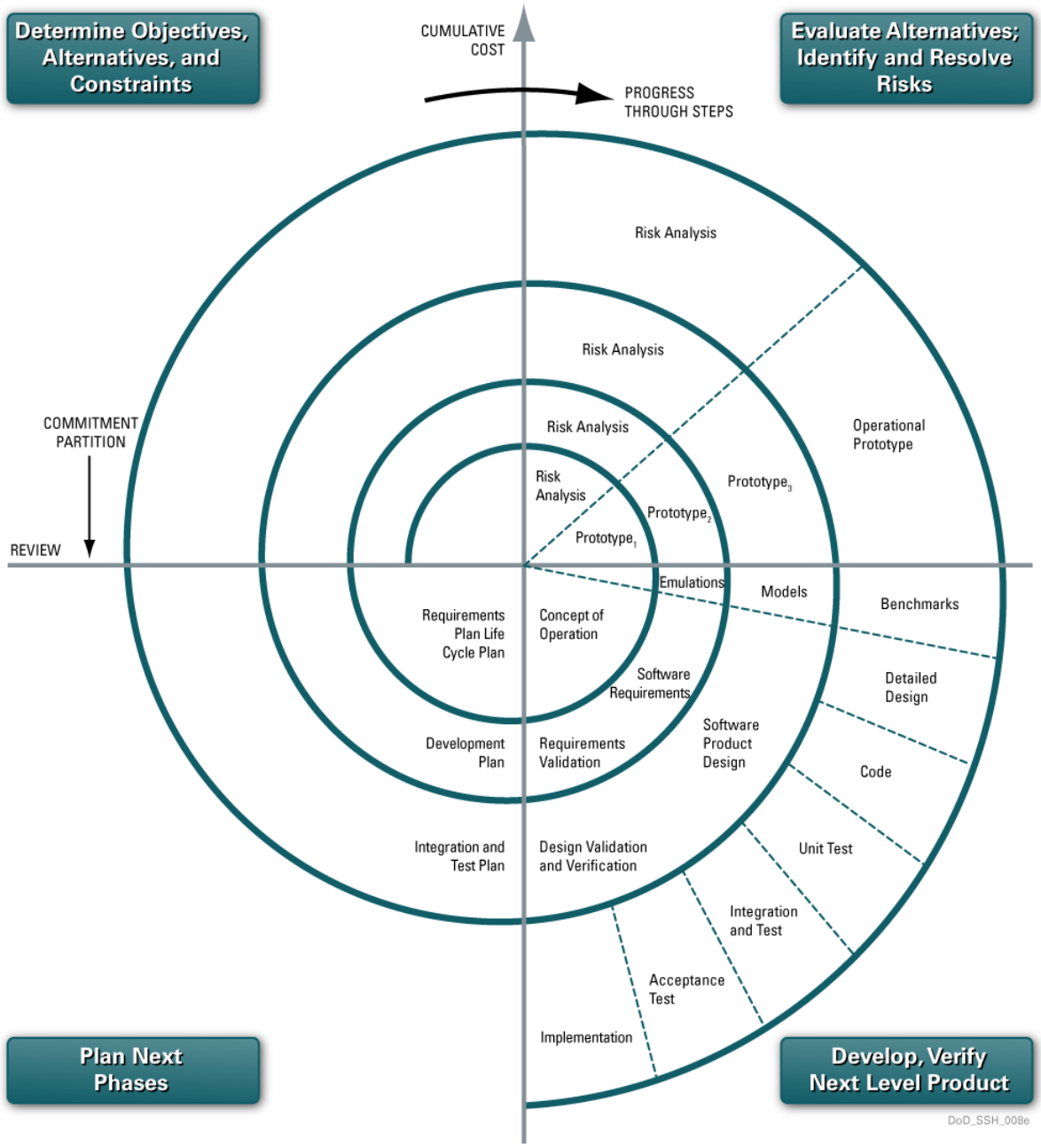

Figure 2-7: Spiral Software Acquisition Lifecycle Model

The Evolutionary Spiral Model, considered for some procurements where Ada language is used, provides an environment that combines model and tool environments and offers the ability to have continual "touch-and-feel" of the software product (as opposed to paper reports and descriptions). This model represents a demonstration-based process that employs a top-down incremental approach, resulting in early and continuous design and implementation validation. Advantages of this approach are that it is built from the top down; it supports partial implementation; the structure is automated, real, and evolved; and each level of development can

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

be demonstrated. Each build and subsequent demonstration validates the process and the structure to the previous build.

2.5.3.2.4 Object-Oriented Analysis and Design

Object-Oriented Analysis and Design is currently the most common technique for designing systems at all levels of abstraction. OOA&D models a system as a collection of objects with specific properties and defined interactions between other objects and the "actors" (e.g., users and external systems). Each object is a model of a fundamental problem-domain concept.[5] Designers identify the necessary control functions external to the system or the system software via use cases, system sequence diagrams, and system operation contracts, then proceed to realize each system operation via the collaboration between object instances derived from the object models.

OOA&D allows the system to be modeled at various levels of abstraction, from the conceptual system level to the software module level where it is an easy jump from the model to the code. Developers can apply object-oriented design principles in conjunction with any of the techniques identified above by defining the objects in a manner consistent with the design technique and the design phase. The Unified Modeling Language (UML) provides a set of notations for the OOA&D of systems. Although these notations lack formal semantic definition, many software engineers consider them a formal design methodology.

The process for software systems safety requires some paradigm shifts when applied to systems developed using OOA&D, especially when the goal is to develop reusable software objects. However, the fundamental system safety and software system safety requirement must still be met. Detailed treatment of the differences in the process is beyond the scope of this Handbook, but is expected to be addressed in the System Safety Program Plan (SSPP). Safety requirements take the form of safety contracts, similar to the System Operation Contracts noted above, which specify requirements that the software classes and objects must not violate. Subsequent analysis of the models verifies that the safety contracts remain in place down to the code development. The analyst must also verify that the resultant code achieves the intent of the safety contracts. The process is more difficult due to the lack of visibility in the resultant code (i.e., there is no possibility to perform code-level analysis or detailed testing at the unit level). This inhibits the safety team's ability to develop safety-specific tests, especially mutation, fault insertion, and failure modes testing.

2.5.3.2.5 Component-Oriented and Package-Oriented Design

Component-Oriented Design and Package-Oriented Design are extensions of the OOA&D methodology to the next level of abstraction. Component-Oriented Design is useful when integrating existing software components or subsystems into a new or improved system. The technique allows developers to take advantage of the full functionality of the existing components and subsystems by designing software that performs necessary functional and data

5 Douglass, Bruce Powell; Doing Hard Time; Addison Wesley; 1999.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

management, accepting the functional and data outputs, and acting on their results. However, application of this process requires full insight into the functionality of the existing components.

Package-Oriented Design extends the process up to the next level and is useful for SoS where developers are designing a super system to control or integrate existing systems or subsystems. This method treats the existing systems as objects and actors within the super system and provides the necessary control and interface to the systems.

The safety team has even less visibility into the software developed using Component or Package-Oriented Design than they do with OOA&D. If reused software components do not already implement the requirements resulting from the safety contract, it is unlikely that the safety team will be able to sufficiently affect the design to mitigate the associated risk. However, it is possible to specify certain attributes of components used in systems developed using Component or Package-Oriented Design, assuming that there are multiple components available from which to select. The safety team can specify those attributes required to maintain an acceptable level of risk in the interaction of a component or system in the new system. If there are available components or packages that have those attributes and also achieve the attributes required by the other "ilities," the designers can incorporate those into the resultant system. However, it is difficult to anticipate what attributes a system may require when the system is undefined. This occurs during the development of a library of reusable components when no system is under development.

Modeling and simulation are often the only means available to analyze the interaction of components in a system developed using Component or Package-Oriented Design. The models and simulators must possess a sufficient pedigree and proven accuracy to allow the safety team to achieve their objectives. The models and simulators must also be accurate enough to react in the same manner as the actual component or system. Because they are used to validate safety, the models and simulators must be classified as safety-critical themselves and be under configuration control and ultimately a contract deliverable.

2.5.3.2.6 Extreme Programming

Extreme Programming, often called Pair-Wise Programming, is a technique used to develop software quickly without high reliability or integrity requirements, such as for video game development. This method is almost a reversion to the code, test, and fix method of software development. Programmers receive a set of requirements, generally in the form of test requirements. One programmer specifically develops the code to pass the test. The other member of the pair questions the interpretation and implementation of the requirements. The programming pair performs testing on the unit using the guidelines provided to them and then provides the tested product to the integration team.

In general, documentation of software developed using Extreme Programming is almost non-existent or is limited to requirements documents and version description documents (code). Extreme Programming is not a recommended practice for safety-significant software. If used, one of the team members must have detailed training in software safety and must have a

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

comprehensive knowledge of the interaction of the module under development with the other modules in the system so that the leaps between and among "phases" are manageable. In addition, the safety team must have a complete understanding of the total system and its interactions between and among the systems-of-systems.

Even with these requirements met, it is likely that the resulting software will contain errors that result in hazardous conditions. A strong negative testing process must be implemented to ensure that the program executes as expected when it encounters unexpected values or conditions. The value is that the working model will be good enough that stakeholders see, understand, and can approve the functionality. The assumption that speed to testing is a priority over completeness or quality assumes that the testing phase will adapt the problems to an acceptable level of quality. Therefore, the end step is always to collect and document the safety capabilities, requirements, mitigations, and proofs (analyses, tests, etc.). These attributes of Extreme Programming basically discount the process as a valid model for safety-critical systems. The costs associated with bringing the process into compliance with safety and security standards generally outweigh the value of the attributes the model possesses. The majority of the additional costs would be associated with requirements for documentation, requirements traceability, and regression testing required for the sustainment of military systems.

2.5.3.3 The Integration of Hardware and Software Lifecycles

A structured lifecycle, complete with controls, audits, reviews, and key decision points, provides the basis for sound decision making based on knowledge, experience, and training. The lifecycle is a logical flow of events representing an orderly progression from a user need to finalized activation, deployment, and support.

The systems approach to software safety engineering must support a structured, well-disciplined, and adequately documented system acquisition lifecycle model that incorporates both the hardware development model and the software development model. Program plans (as described in Appendix C.9) must describe how each applicable engineering discipline will interface and perform within the development lifecycle. Graphical representations of the lifecycle model of choice for a given development activity must be provided during the planning processes to aid in the planning and implementation processes of software safety engineering. This also allows for the integration of safety-significant requirements and guidelines into the design and code phases of software development. This approach will further assist in the timely identification of safety-specific test and verification requirements to prove that original design requirements have been implemented as intended. This process further allows for the incorporation of safety inputs to the prototyping activities in order to demonstrate safety concepts.

2.5.4 A Team Solution

System safety engineers cannot reduce the safety risk of systems software by themselves. The software safety process must be an integrated team effort between the engineering disciplines. As shown in Figure 2-1, software, safety, and systems engineering are pivotal players on the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

team. The management block is analogous to a conductor that provides the necessary motivation, direction, support, and resources for the team to perform as a well-orchestrated unit.

It is the intent of this Handbook to demonstrate that neither the software developers nor safety engineers alone can accomplish the necessary tasks to the level of detail required. This Handbook will focus on the required tasks of the safety engineer, the software engineer, the software safety engineer, the system engineers, the design engineers, and the interfaces between them. Regardless of who executes the individual software safety tasks, each engineer must be aware of the duties, responsibilities, and tasks required from each functional discipline. Each must also understand the time (in terms of lifecycle schedule), place (in terms of required audits, meetings, and reviews), and functional analysis tasks to be produced and delivered at any point in the development process. Section 4 will expand on the team approach, including the necessary planning, process tasks, products, and risk assessment tasks. Figure 2-8 uses a puzzle analogy to demonstrate that the software safety approach must establish integration between functions and among engineers. Any piece of the puzzle that is missing from the picture will propagate into an unfinished or incomplete—and potentially hazardous—software safety effort.

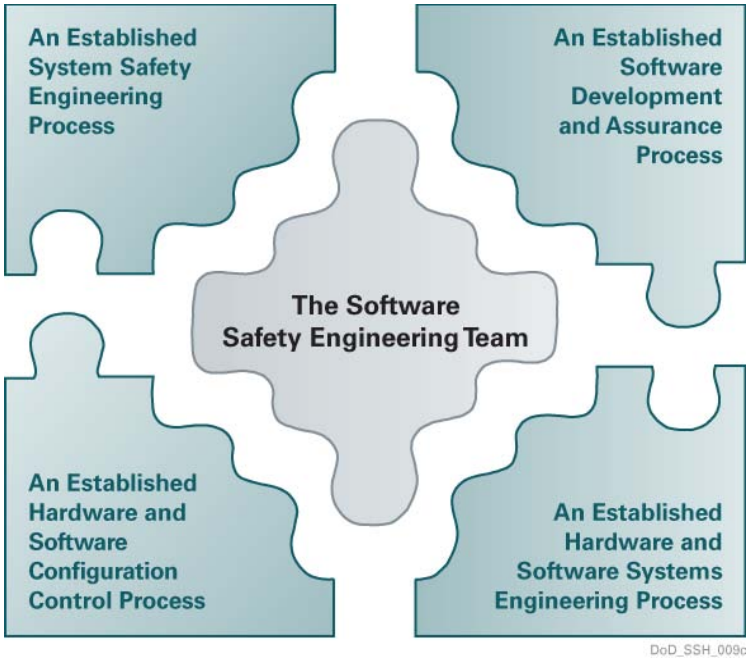

Figure 2-8: Integration of Engineering Personnel and Processes

The elements contributing to a credible and successful software safety engineering program will include:

- A defined and established system safety engineering process
- A structured and disciplined software development process

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- An established hardware and software systems engineering process
- An established hardware and software configuration control process
- An established software assurance and integrity process for safety-critical software development and testing
- An established software system safety engineering hazard analysis process
- An integrated SSS team responsible for the identification, implementation, and verification of safety-specific requirements in the design and code of the software.

2.5.5 Systems of Systems Hazards and Causal Factors

2.5.5.1 Safety as a System Property

Safety is a property of a system, not a property of the components (including software) that comprise the system. When the context of the system changes, the safety properties also change, including those attributes, interlocks, and checks and balances designed to mitigate the risks associated with the system. Integrating systems into a SoS can create new hazards and hazard causal factors. The occurrence of these hazards and hazard causal factors results from the interface and interaction between the subsystems or systems (interface-related), functions of the integrated systems (functional), or hazards and hazard causal factors created by the proximity of the systems (zonal hazard causes). The advantages of the SoS derive not only from the ability to share data between the systems, but the synergy that occurs when the systems are integrated. However, that synergy itself can introduce hazards and hazard causal factors that require mitigation at either (or both) the SoS or system level.

2.5.5.2 Functional Hazard Causal Factors

Functional hazard causal factors result from new, expanded, or modified functionality created by the integration of the systems or subsystems. In general, functional hazard causes are a subset of interface hazard causal factors in the SoS context. However, traditional Functional Hazard Assessment techniques will not identify the associated hazards and hazard causal factors, especially in a complex SoS.

An example of a simple functional hazard is the integration of a Weapon Control System (WCS) with a Command and Control System (CCS), especially where the WCS was a stand-alone system. The WCS accepts targeting data and commands from the CCS as being from a trusted source. The WCS does not possess the fidelity or functionality to determine whether the commands from the CCS are valid. As the complexity of the CCS increases, especially when multiple Command and Control Systems may direct the WCS (as may occur in a federated SoS), the likelihood of introducing functional hazards and hazard causal factors increases. The System Hazard Analysis (SHA) for the WCS (if performed) will likely identify many of the functional hazard causal factors related to the interface between the CCS and the WCS.[6] However, if that analysis was not performed, as may be the case where the WCS is an older stand-alone system

6 This assumes that the original WCS was designed with a CCS interface.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

now being integrated into a federated SoS, the safety team must perform an analysis to identify the hazard causal factors that arise from the integration.

In this scenario, it is likely that someone other than the original WCS development team is designing the interface between the CCS and the WCS. These designers may not have the detailed knowledge of the WCS functionality and implementation to develop a robust and safe interface with the CCS. Likewise, the safety team may be constrained to analyze only the newly developed aspects of the WCS. The hazard causal factors identified in the WCS SHA will affect the design and implementation of every CCS[7] tied into the federated SoS, as well as systems in the SoS that may not have control functionality over the WCS. An example of the latter is a sensor system providing data to the federated SoS. If the units of measure or format are not completely compatible with the WCS (or SoS), a hazard may result. Likewise, the coordinate reference system and timeliness of the data (such as the time delay when a "hostile" is reclassified as a "friendly") will affect the safety risks associated with the interaction between the WCS and the SoS. It is imperative that SoS hazard analysis includes the entire SoS in the context of all that is interfaced functionally and physically together, not just the "deltas" between them.

2.5.5.3 Interface-Related Hazard Causal Factors

Interface and interaction hazard causes result from data or controls passed between systems or subsystems. Control-related hazards and hazard causal factors generally fall into the functional hazard causal factor category, although there are exceptions (e.g., real-time constraints). Interface and interaction hazard causes (including data senescence) can be very complex yet subtle, difficult to identify, and difficult to mitigate, especially with systems low in the hierarchy. Identifying these interface-related causes and designing test cases to validate they have been mitigated can be complex. Adding to the complexity are the iterative meetings with various stakeholders and developing the necessary models and simulators to prove their existence.

2.5.5.4 Zonal Hazard Causes

Zonal hazard causes result from the operation or failure of a system or subsystem that directly affects the safe operation of another system even though the systems do not have a direct interface. An example of a zonal hazard cause is an aircraft's electrical system cable in close proximity to the aircraft's hydraulic control lines. A failure in either could adversely affect the other, resulting in loss of both systems. In this case, loss of the aircraft is likely. Zonal hazard causes in a SoS can be directly identifiable or very difficult to identify, even transitory as with systems that pass by each other at very high speeds. In general, zonal hazards are difficult to identify or control because the impact of the consequences of one system over another may be transparent to a high-level user. One example is the control of a missile launch by a Forward Operations Command Center (FOCC) from a ship in a battlegroup. That launch could create a hazardous condition on another ship (e.g., separated missile booster falling onto another ship) if the launching platform does not have some level of control of the launch process. Conversely,

7 This is especially true if the WCS update is unable to mitigate all of the identified risks.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

the FOCC could have the information necessary to determine that launching from one platform may result in a hazard to another and may select an alternate platform. However, this capability adds significant complexity to the FOCC functionality and, in order to develop it, the FOCC designers must be aware of all hazardous conditions that may exist.

2.5.5.5 Data Interfaces

Data interfaces within the SoS are especially prone to contributing to hazardous conditions and hazard causal factors. Individual systems within the SoS may use different reference systems (e.g., relative coordinates vs. true coordinates), different units of measure (English vs. metric), and different data rates and formats. Systems can also use different data protocols (e.g., Internet Protocol Version 4 vs. Internet Protocol Version 6 vs. Variable Message Format), message formats, and message structures. Security protocols and data encryption algorithms may be different. One system may require embedded cyclic redundancy check (CRC) checksums that another system cannot tolerate or accommodate.

The SoS must accommodate all of these differences or the individual systems must change to match a common data structure for the SoS. The latter is unlikely because the cost of modifying the individual systems is often prohibitive. Therefore, the SoS must have the capability to determine which system(s) is providing data, which system(s) receives that data, and the compatible formats for both systems. The SoS must determine which data the receiving system requires and which data it cannot accommodate and ensure that it receives that data. The SoS must also ensure that the data rate is compatible with the receiving system. If the sending system does not provide all of the data necessary to the receiving system, the SoS must find another source for the missing data. The SoS must also determine the necessary translation from the sending system's reference system and unit of measure to the receiving system's reference system and unit of measure. As new or modified systems enter the SoS, the system must change to accommodate the additional data infrastructure without adversely impacting the operation of the other systems within the SoS. The result is that the data infrastructure within the SoS becomes complex and subject to processing delays due to the necessary translation. Coupled with the fact that many SoS will operate over large areas, data senescence can be a significant hazard causal factor within the SoS. The SoS must also ensure that the acknowledge/not acknowledge message back to the sending system is expected, understood, and acted upon accordingly.

The military's requirements to identify and catalog Interface Exchange Requirements (IERs), along with an Information Support Plan (ISP), are key to identifying the data exchange requirements of a system or SoS. Supporting analysis will include what data is produced, who will use the data, and how it will be exchanged. Data exchange issues affect many individual systems already in service although they were designed to the same Interface Requirements Specifications. In a complex SoS, the likelihood that a data exchange issue will adversely affect the safety of the SoS increases as the number of systems increases. Many systems will share common resources in the SoS. If an operation requires significant use of a particular type of system with limited availability (e.g., the Airborne Warning and Control System) or a system must perform multiple tasks in the SoS context (e.g., a radar required to perform both volume

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

search and missile guidance), the resource sharing may result in a variety of hazard causal factors, including data senescence issues. Resource conflicts, bandwidth limitations, and data senescence can all result in the loss of critical data and could adversely affect safe operations of the SoS even though the elemental systems may have had the benefit of high-quality safety programs.

2.5.5.6 COTS

As with other systems under development, a SoS may use many commercial off-the-shelf (COTS) products, including computers and software, communications devices, and a variety of other commercially-developed products. The safety issues associated with COTS in safety-significant systems increases with the complexity of the SoS.

COTS vendors do not usually create the software to be used in safety-significant systems or safety-critical applications, and therefore may not conform to safety guidelines or testing. COTS products are often black box and lack complete or adequate documentation of the establishment of safety requirements for the system. Therefore, COTS may require significant analysis and testing in the context of the safety criticality of the system. Treating COTS products as black boxes is also necessary when the vendor cannot or will not supply a copy of the source code to the user.

The subject of COTS from a system and software safety perspective is further detailed in Appendix D of this Handbook.

2.5.5.7 Technology Issues

Technology obsolescence and technology refresh are significant issues within modern systems, especially those using COTS hardware or software. If not properly analyzed and tested, any technology insertion can adversely impact the modified system. Determining the effects of that insertion at the SoS level can be challenging. Technological issues that impact systems are compounded in the SoS environment, primarily due to the effect those issues have on the individual systems within the SoS. Modifications at the SoS level may be required to accommodate the updated system; however, the SoS will still need to interface with and control the systems of the same family that have not received the update. This issue is multiplied by the number of elemental systems.

2.6 Handbook Organization

This Handbook is organized to provide the capability to review or extract subject information important to the reader. For example, the Overview may be the only portion required by the executive officer, program director, or PM to determine whether a software safety program is required for their program. The Overview will provide the necessary motivation, authority, and impetus for establishing a software safety program consistent with the nature of development.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Engineering and software managers will need to read further into the Handbook to obtain the managerial and technical process steps required for a software-intensive, safety-significant system development program. Safety program managers, safety engineers, software engineers, systems engineers, and software safety engineers will need to read even further to gather the information necessary to develop, establish, implement, and manage an effective SSS Program. This includes the "how-to" details for conducting various analyses required to ensure that the system software will function within the system context with an acceptable level of safety risk. Figure 2-9 depicts the layout of the four sections of the Handbook, the appendices, and a brief description of the contents of each.

As shown in Figure 2-9, Section 1 provides an Overview of the Handbook and the subject of software safety. Section 1 also communicates the requirement and authority for an SSS program; motivation and authority for the requirement; and the roles and responsibilities of the customer, the program, and the design and development engineering disciplines.

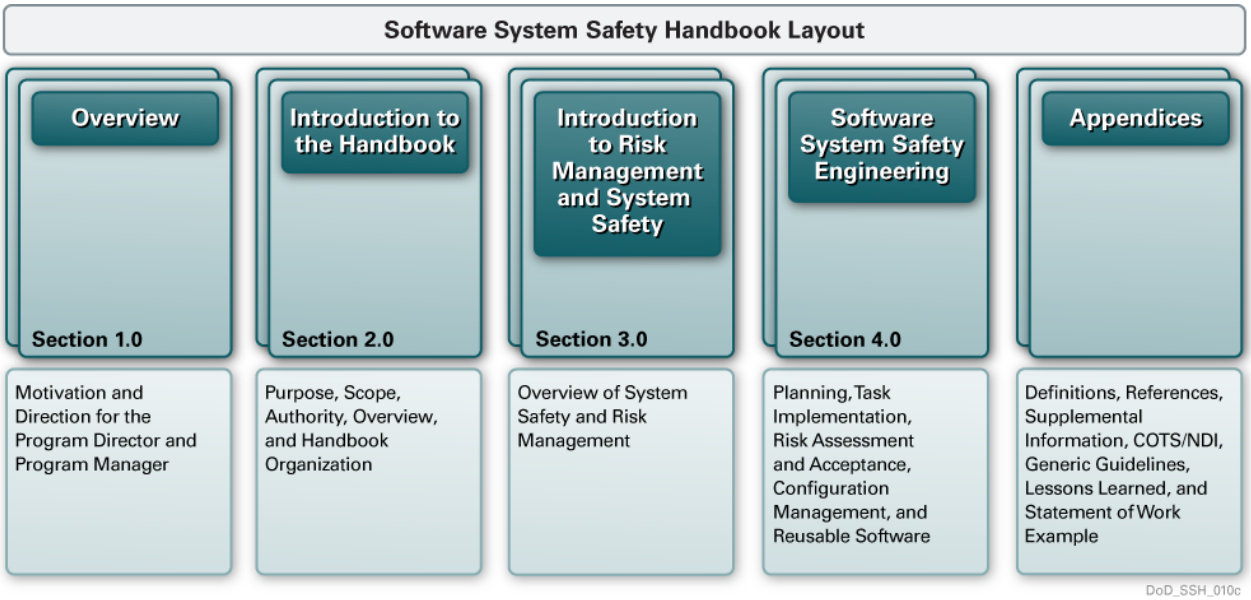

Figure 2-9: Handbook Layout

Section 2 provides an Introduction, including an in-depth description of the purpose and scope of the Handbook; the authority for the establishment of an SSS program on DoD procurements and acquisition research and development activities; national and international standards related to software safety; and issues and concerns associated with modern system development. Section 2 also provides a description of the layout of the Handbook as it applies to the acquisition lifecycle of a system development.

Section 3 is an introduction to system safety engineering and management for those readers not familiar with the MIL-STD-882 methods and the approach for establishment and implementation of a System Safety Program (SSP). This section provides an introduction to risk management

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

and how safety risk is an integral part of the risk management function. Section 3 also provides an introduction and an overview of the systems acquisition, systems engineering, and software development processes and guidance for the effective integration of these efforts in a comprehensive systems safety process.

Section 4 provides the "how-to" of a baseline software safety program. Not all acquisitions and procurements are similar, nor do they possess the same problems, assumptions, and limitations in terms of technology, resources, development lifecycles, and personalities. This section provides guidance for careful planning and the forethought required to establish, tailor, and implement an effective SSS program. Figure 2-10 provides the reader with the steps required for planning, task implementation, and risk assessment and acceptance for an SSS program. Appendix C and Appendix D provide information regarding the management of configuration changes and issues pertaining to software reuse and COTS software packages.

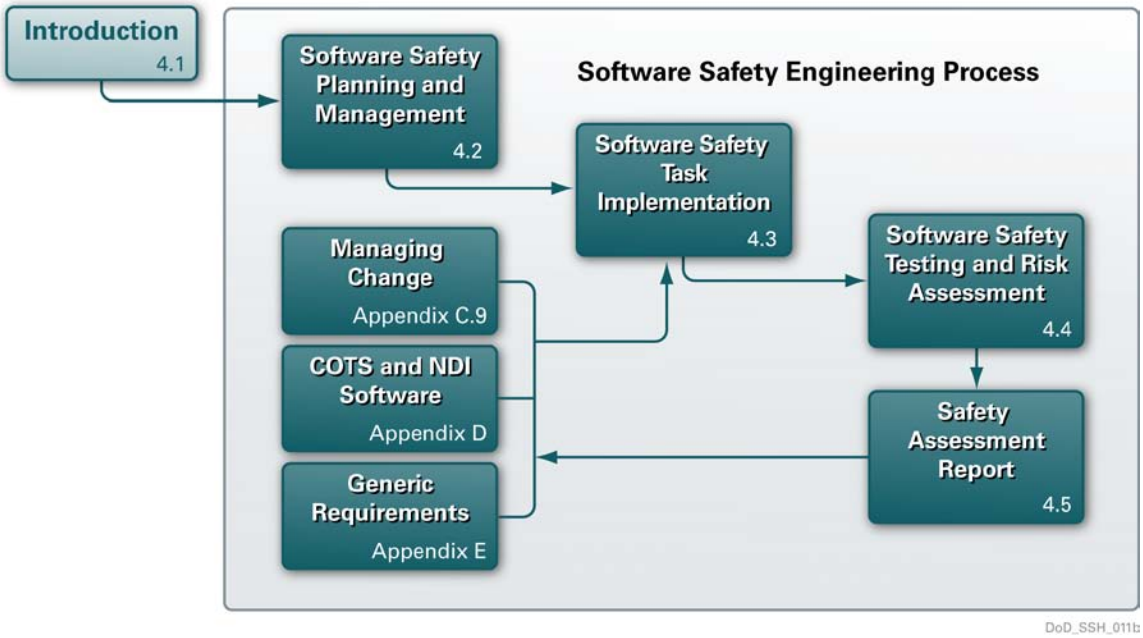

Figure 2-10: Section 4 Format

2.6.1 Planning and Management

Section 4.2 begins with an introduction and transitions to the planning and management required for establishing an SSS program, program interfaces, contractual interfaces and obligations, safety resources, and program plans. This section assists and guides the safety manager and engineer in the required steps of software safety program planning. Although some subject areas may not be required for individual product procurements, the planning process should address and consider each area. With supporting evidence and rationale, it is acceptable to determine that a specific activity or deliverable is not appropriate or necessary for an individual program.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Section 4 also addresses the metrics necessary to measure the effectiveness of the software safety program, the guidelines for managing change both during and after the software safety program is complete, and the tasks necessary to address commercially-developed and other non-developmental software incorporated into the system design. Be advised that a software safety program is only "complete" when the software is taken out of service as the program is still necessary during the maintenance and support phase of the program.

2.6.2 Task Implementation

Task implementation is the heart of the Handbook as applied to implementing a credible software safety program. Task implementation establishes a step-by-step baseline of best practices for reducing the safety risk of software performing safety-significant functions within a system. These process steps are not always serial in nature. Although presented in a serial format for ease of reading and understanding, many activities will require parallel processing and effort from the safety manager and engineer. Activities as complex and interface-dependent as software development within a systems acquisition process will seldom have required tasks line up where one task is complete before the next one begins. This is often apparent in the development of an SSS program and milestone schedule (see Section 4.3)

This section of the Handbook describes the tasks associated with contract and deliverable data development (including methods for tailoring), safety-critical function identification, preliminary and detailed hazard analysis, safety-specific requirements identification, implementation, test and verification, and residual risk analysis and acceptance. This section also includes the participation in trade studies and design alternatives.

2.6.3 Residual Safety Risk Assessment and Acceptance

The risk assessment and acceptance portion of Section 4 (Section 4.4) focuses on identifying residual safety risk in the design, test, and operation of the system. This section includes the evaluation and categorization of hazards and their impact on operations, maintenance, and support functions. This section also includes the graduated levels of programmatic sign-off for hazard and failure mode records of subsystem, system, and operations and support hazard analyses. This section details the tasks required to identify the hazards remaining in the system; assess safety risk impact with severity, probability, or software control criticality; and determine the residual safety risk.

2.6.4 Supplementary Appendices

The Handbook appendices include acronyms, definitions of terms, references, supplemental system safety information, generic safety requirements and guidelines, lessons learned from previous systems and safety programs, metrics for measuring the effectiveness of a software safety program, and other elements pertaining to the accomplishment of SSS tasks.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

3 Introduction to Risk Management and System Safety

3.1 Introduction

SSS team members who are not familiar with system safety and those who need to be more familiar with the concept of the Mishap Risk Index and how hazards are rationally assessed, analyzed, correlated, and tracked should read this section because Section 3 discusses:

- Risk and its application to the software system safety program
- Programmatic risks
- Safety risk management.

3.2 A Discussion of Risk

There are risks associated with every action we attempt in our lives. Some risk is easy to identify and some can easily be overlooked. Some risk possesses substantial hazard potential while some risk is considered insignificant. In our society, there are times when taking risk is considered foolhardy, irrational, and something to be avoided. In addition, risk imposed on us by others is generally considered to be unacceptable. Risk is an unavoidable part of our everyday lives.

The risk to be evaluated in this Handbook primarily pertains to safety risk and the mishap risks associated with designing, testing, producing, operating, supporting, and decommissioning systems. Realistically, some mishap risk potential must be accepted. Systems are never risk free. For example, totally safe aircraft will never fly because the potential for a crash is still possible if it becomes airborne. The residual safety risk in the fielded system is the direct result of the accuracy and comprehensiveness of the system safety program. How much risk is accepted or not accepted is the prerogative of management for an acquisition program.

As tradeoffs are considered and the design progresses, it may become evident that some of the safety parameters are forcing higher program risk. From the Program Manager's perspective, a relaxation of one or more of the safety requirements may appear to be advantageous when considering the broader perspective of cost and performance optimization. The PM may make a decision against the recommendation of the system safety manager. The system safety manager must recognize such management prerogatives. The prudent Program Manager must make the decision whether to resolve the identified issues or formally document acceptance of the risk. When the PM decides to accept risk, the decision must be a coordinated and informed decision and must be communicated with all effected organizations. Risk acceptance should also be documented so that all involved will understand the elements of the decision and why it was made.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Accepting risk is an action of both risk assessment and risk management. The risk assessment process must consider:

- Risk is a fundamental reality
- Risk management is a process of tradeoffs
- Quantifying risk does not ensure safety
- Risk is often a matter of perspective.

3.2.1 Risk Perspectives

When discussing risk, there are three main perspectives:

- Risk exposure to an individual
- Risk exposure to the general public – Society is interested in guaranteeing minimum individual risks for each of its members and is concerned about the total risk to the general public
- Risk exposure to public or private institutions – The institution responsible for an activity can be a private company or a Government agency. From this point of view, it is essential to keep individual risks to employees and others to a minimum. An institution's concern is also to avoid accidents. From an institutional perspective, the results of an accident can be detrimental in terms of loss of facilities, lives, schedule, budget, operational capability, and reputation.

The system safety effort is an optimizing process that varies in scope and scale over the lifetime of the system. The system safety program balances system safety with cost, performance, and schedule. Without an awareness of the system safety balance on the part of both the Program Manager and the system safety manager, they cannot discuss when, where, and how much they can afford to spend on a system safety effort. Decision-makers cannot afford mishaps that will prevent the achievement of the mission objectives, nor can they afford systems that cannot meet operational effectiveness requirements due to overstated safety goals.

3.2.2 Safety Management Risk Review

The SSP examines the interrelationships of all components of a program and its systems with the objective of bringing mishap risk or safety risk reduction into the management review process for automatic consideration in the total program perspective. This process involves the preparation and implementation of system safety plans, the performance of system safety analyses on both system design and operations, and risk assessments in support of both management and system engineering activities. The system safety activity provides the manager with a means of identifying what the risk of mishap is, where a mishap can be expected to occur, and what alternate designs are appropriate. This process also verifies implementation and effectiveness of hazard controls. If left unresolved, engineering and management issues can result in a mishap. When a mishap occurs, then it is no longer a risk, but a safety problem with

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

consequences and program management issues. Identification and control of mishap risk is an engineering and management function. This is particularly true of software safety risk.

3.3 Types of Risk

There are various models describing the risks listed below. The model in Figure 3-1 follows the system safety concept of risk reduction.

- Total risk is the sum of identified and unidentified risks
- Identified risk is that risk which has been determined through various analytical techniques. The first task of system safety is to identify as many potential risks as practical. The time and costs of analytical efforts, the quality of the safety program, and the state of technology impact the amount of risk identified

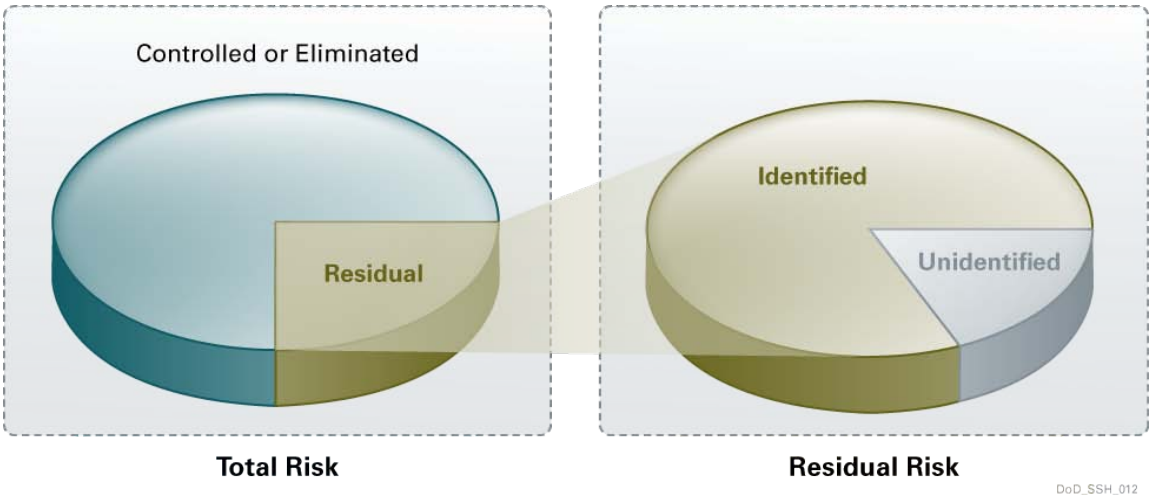

Figure 3-1: Types of Risk

- Unacceptable risk is that risk which cannot be tolerated by the managing activity. Unacceptable risk is a subset of identified risk that is either eliminated or controlled
- Residual risk is the risk remaining after system safety efforts have been fully employed. Residual risk is sometimes erroneously considered to be the same as acceptable risk. Residual risk is actually the sum of unacceptable risk (unmitigated or uncontrolled), acceptable risk, and unidentified risk. This is the total risk passed on to the user that may contain some unacceptable risk
- Acceptable risk is the part of identified risk that is allowed to persist without further engineering or management action and is acceptable by the PM. Acceptable risk is documented and generally leads to a procedural work-around or warnings, cautions,

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

and advisories, depending on the risk level. Acceptable risk is recognized by the managing activity; however, it is the user who is exposed to this risk.

- Unidentified risk is the risk that has not been determined. Unidentified risk is real and important, but it cannot be measured. Some unidentified risk is subsequently determined and measured when a mishap occurs. Some risk is never known.

3.4 Areas of Program Risk

Within DoD, risk is defined as a potential occurrence that is detrimental to plans or programs. While risk includes safety risk as required by DoD mandate, it is not always a one-to-one comparison. Program risk is measured as the combined effect of the likelihood of the occurrence and a measured or assessed consequence given to that occurrence. The perceived risk to a program may be different for program management, systems engineers, users, and safety professionals. The responsibility of defining program risk is usually assigned to a small group of individuals from various disciplines that can evaluate the program risks from the broad perspective of the total program. Safety risk is considered in the SSP and is later "rolled-up" to program risk. Program-based issues include business, cost, schedule, technical, and programmatic considerations. Although the PM may delegate risk management responsibility to an individual group, the successful management of a program's risk is dependent on contribution and input from all individuals involved in program management and engineering design functional activities.

The risk management group is usually assigned to (or contained within) the systems engineering group. This group is responsible for identifying, evaluating, measuring, documenting, and resolving risk within the program, including recognizing and understanding the warning signals that may indicate that the program or elements of the program are off track. The risk management group must also understand the seriousness of the issues identified and develop and implement plans to reduce the risk. A risk management assessment must be made early in the development process, and the risks must continually be reevaluated throughout the development lifecycle. Members of the risk management group and the methods of risk identification and control should be documented in the program's Risk Management Plan (RMP).

Risk management[8] must consist of three activities:

- Risk Planning – This process provides organized, purposeful thought to the subject of eliminating, minimizing, or containing the effects of undesirable events and their consequences.

8 Selected descriptions and definitions regarding risk management are paraphrased from the Defense Systems Management College (DSMC) *Systems Engineering Management Guide*; DSMC; January 1990.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Risk Assessment – This process examines a situation and identifies the areas of potential risk. The methods, techniques, and documentation often used in risk assessment include:
  - Systems engineering documents
  - Operational requirements document
  - Operational concepts document
  - Lifecycle cost analysis and models
  - Schedule analysis and models
  - Baseline cost estimates
  - Requirements documents
  - Lessons learned files and databases
  - Trade studies and analyses
  - Technical performance measurements and analyses
  - Work breakdown structures
  - Project planning documents
- Risk Analysis – This process determines the probability of events and the potential severity consequences associated with these events relative to the program. The purpose of a risk analysis is to discover the causes, effects, and magnitude of the potential risks and develop and examine alternative actions that could reduce or eliminate these risks. Typical tools or models used in risk analysis include:
  - Schedule network model
  - Lifecycle cost model
  - Quick reaction rate and quantity cost impact model
  - System modeling and optimization.

Although safety, by definition, is a part of technical risk, it can impact all areas of programmatic risk, as described in subsequent sections.

3.4.1 Schedule Risk

The master systems engineering and software development schedule for a program contains numerous areas of programmatic risk, such as schedules for new technology development, funding allocations, test site availability, critical personnel availability, and rotation. Each of these elements has the potential for delaying the development schedule and can induce unwarranted safety risk to the program. While these examples are not the only sources of schedule risk, they are common to most programs. The risk manager must identify, analyze, and control risks to the program schedule by incorporating measures into the planning, scheduling, and coordinating activities to minimize impact to the development program.

To help accomplish these tasks, the systems engineering function maintains the Systems Engineering Master Schedule (SEMS) or Integrated Master Schedule and the Systems Engineering Detailed Schedule (SEDS). Maintaining these schedules helps guide the interface between the customer and the developer, provides the cornerstone of the technical status and reporting process, and provides an interface between engineering disciplines and respective system requirements. An example of the integration, documentation, tracking, and tracing of

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

risk management issues is depicted in Figure 3-2. Note that the SEMS and SEDS schedules and the risk management effort are supported by a risk issue table and risk management database. These tools assist the risk manager in the identification, tracking, categorization, presentation, and resolution of managerial and technical risk. In most DoD programs, these are included in the Systems Engineering Plan (SEP) required for approval at each milestone by the MDA.

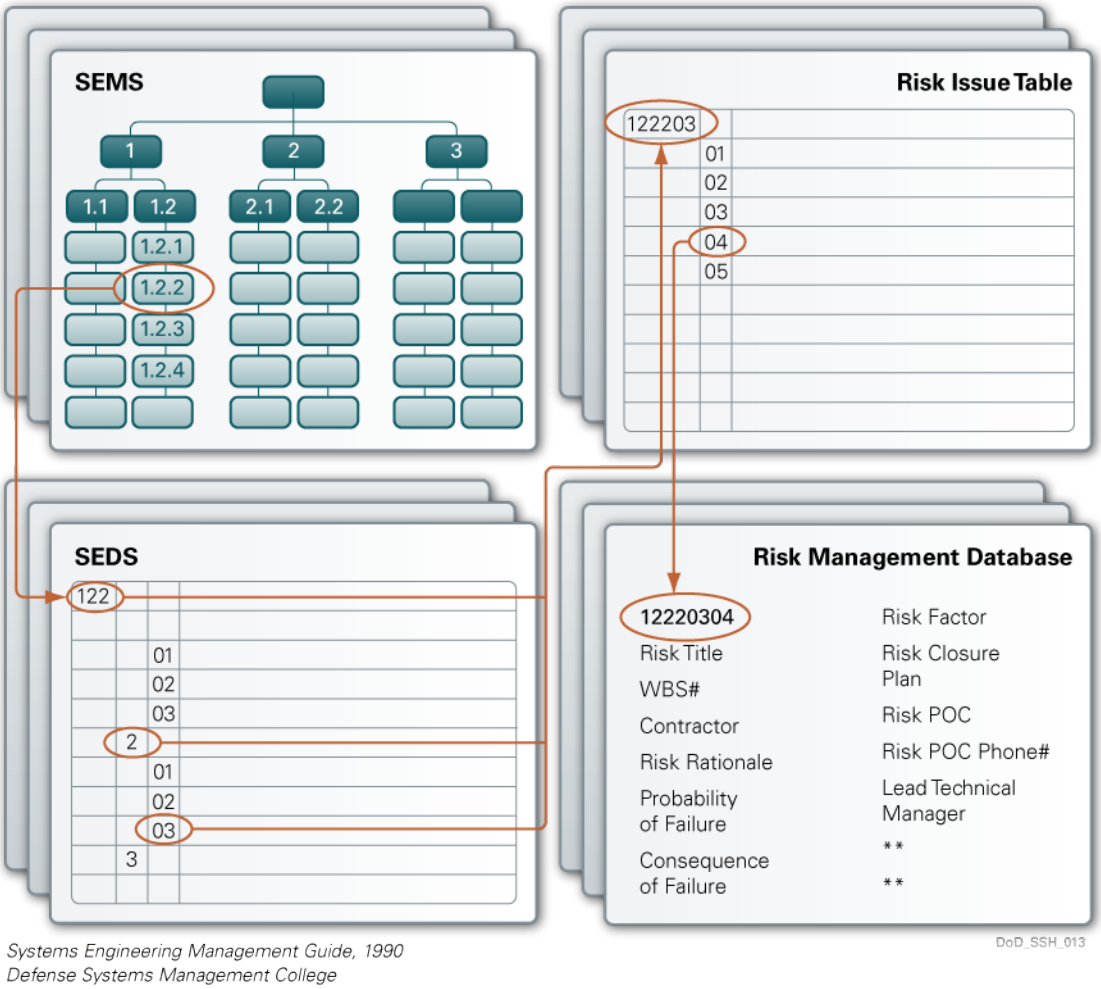

Figure 3-2: Systems Engineering and Risk Management Documentation

Software developers for DoD customers or agencies have been found lacking in systems engineering and planning, as demonstrated by the 2006 Office of Secretary of Defense Research Development Test and Evaluation Project Justification funding request which stated, "Shortcomings in software development often lead to schedule slippage, cost growth, and mission compromises." Schedule risk is an important consideration of a software development program. The schedule can become the driving factor, forcing the delivery of an immature and inadequately tested critical software product to the customer. The risk manager, in concert with

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

the safety manager, must ensure that the delivered product does not introduce unacceptable safety risk to the user, system, maintainer, or the environment.

Implementation of a SwSSP and safety requirements early in the software design process produces a risk reduction schedule by decreasing the potential for re-design and re-code of software possessing safety deficiencies. Be aware that most Work Breakdown Structure (WBS) and schedule relationships limit Government oversight to Level 3 on the WBS, and both software and safety risk issues are typically below Level 5 which is not visible in most contract reports. For example, in Figure 3-2, the "call-out" is at Level 3 and risk is being managed at Level 3. Unless system and software safety risk is recast at the system level by the safety professional, the identified risk may not be communicated to the appropriate levels of management.

3.4.2 Budget Risk

Schedule risk goes hand-in-hand with budget risk. Although these risks can be mutually exclusive, this is seldom the case. The lack of monetary resources is always a potential risk in a development program. Within DoD research, development, and acquisition agencies, there is always the potential for budget cuts or Congressionally-mandated program reductions. Considering this potential, budgetary planning, cost scheduling, and program funding coordination are paramount to the risk management team. The team must ensure that budgetary plans for current and out years are accurate and reasonable, and that potential limitations or contingencies for funding are identified, analyzed, and incorporated into the program plans. Unless the contract is unique, the Government insight into budgets is limited to Level 3 within the WBS. The software systems safety engineer can help by linking risk up to Level 3 of the WBS object as a system-level effect.

In system safety terms, the development of safety-critical software requires significant program resources, highly skilled engineers, increased training requirements, software development tools, modeling and simulation, and facilities and testing resources. To ensure that this software meets functionality, safety, and reliability goals, these activities become drivers for both the budget and schedule of a program. Therefore, the safety manager must ensure that all safety-specific software development and test functions are prioritized in terms of safety risk potential to the program and to the operation of software after implementation and are traceable to the highest levels of the WBS and schedule items. The prioritization of safety hazards and failure modes, requirements, specifications, and test activities attributed to software helps facilitate and support the tasks performed by the risk management team, incorporating activities necessary to minimize the safety risk potential for the program.

3.4.3 Sociopolitical Risk

Sociopolitical risk can be a challenge from a risk management perspective. Sociopolitical risk is predicated more on public and political perceptions than on absolute fact. Examples of this type of risk can be seen during the design, development, test, and fielding of a nuclear weapon system

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

in a geographical area that has a strong public or political resistance. With this example in mind, several programmatic areas become important for discussion. Program design, development, and test results have to be predicated on complete and technical facts. This will preclude any public perception of attempts to hide technical shortfalls or safety risk. Social and political perceptions can generate programmatic risk that must be considered by the risk managers. This includes the potential for budget cuts, schedule extensions, or program delays due to funding cuts as a result of public protest and influence on politicians.

Safety plays a significant role in influencing sensitivities toward a particular program. Safety must be a primary consideration in assessing risk management alternatives. In the nuclear weapon system example, if an accident (even a minor accident without injury) occurs during testing, it could result in significant political repercussions and perhaps program cancellation.

Sociopolitical risk may also change during the lifecycle of a system. For example, explosive handling facilities once located in isolated locations may be encroached upon by residential areas. Protective measures adequate for an isolated facility may not be adequate as residents, especially those not associated with the facility, move closer. While the Program Manager cannot control the later growth in population, they must consider this and other factors during the system development process.

The act of documenting risk and communicating that risk to the appropriate level of management is sometimes problematic. It can either be perceived as a “program killer” or an issue that does not possess the adequate resources to solve. This can in turn lead to push-back to even document the high risk items. It is the responsibility of the safety engineer and the PM to inform the risk acceptance authority and senior decision-makers of all risk.

3.4.4 Technical Risk

Technical risk is where safety risk is most evident in system development and procurement. Technical risk is the risk associated with the implementation of new technologies or new applications of existing technologies into the system being developed. These include the hardware, software, human factors interface, and environmental safety issues associated with the design, manufacture, fabrication, test, deployment, operation, and maintenance of the system. Technical risk results first from the requirements themselves (build a weapon), and then from poor identification, incorporation, and integration of system performance requirements that meet the intent of the user and system specifications while remaining safe. The inability to incorporate defined requirements into the design (e.g., lack of technology base, funds, and experience) increases the technical risk potential.

Systems engineers are tasked with activities associated with risk management and are assigned the responsibility of requirements management within the design engineering activities. The systems engineering function includes specification development, functional analysis, requirements allocation, trade studies, design optimization and effectiveness analysis, technical

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

interface compatibility, logistic support analysis, program risk analysis, engineering integration and control, technical performance measurement, and documentation control.

In terms of software safety, the primary objective of risk management is to understand that safety risk is a part of the technical risk of a program and to relate that risk to the appropriate management authority in terms commonly used by risk managers. However, a safety-significant mishap or accident will have a negative consequence on the budget, schedule, sociological. and programmatic areas of total risk management. Therefore, all program risks must be identified, analyzed, and eliminated or controlled. This includes safety risk, and thus, software safety risk.

3.5 System Safety Engineering

To understand the concept of system safety as it applies to software development, the user needs a basic introduction and description of system safety because software safety is a subset of the system safety program activities. "System safety as we know it today began as a grassroots movement that was introduced in the 40s, gained momentum during the 50s, became established in the 60s, and formalized its place in the acquisition process in the 70s. The system safety concept was not the brain child of one person, but rather a call from the engineering and safety communities to design and build safer equipment by applying lessons learned from our accident investigations."[9]

System safety grew out of "conditions arising after World War II when its parent disciplines, systems engineering and systems analysis, were developed to cope with new and complex engineering problems."[10] System safety evolved in conjunction with systems engineering and systems analysis. Systems engineering considers "the overall process of creating a complex human/machine system and systems analysis providing the data for the decision-making aspects of that process and an organized way to select among the latest alternative designs."[11]

In the 1950s, political pressure focused on safety following several catastrophic mishaps such as Atlas and Titan Intercontinental Ballistic Missiles exploding in silos during operational testing. Investigation into the cause of these accidents revealed that a large percentage of causal factors could be traced to deficiencies in design, operation, and management that should have been detected and corrected prior to placing the system in service. This recognition led to the development of system safety approaches to identify and control hazards in the design of the system to minimize the likelihood and severity of first-time accidents.

As system safety analytical techniques and managerial methods evolved, they have been documented in various Government and commercial standards. The first system safety specification was created by the Air Force in 1966, Military Specification - 38130A. In June 1969, MIL-STD-882 replaced this standard, and a system safety program became mandatory for

9 *Air Force System Safety Handbook*, August 1992.
10 Leveson, Nancy G.; *Safeware, System Safety, and Computers*; Addison Wesley; 1995; page 129.
11 Ibid, page 143.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

all DoD-procured products and systems. Many of the later system safety requirements and standards in industry and other Government agencies were developed based on MIL-STD-882, and remain so today. As DoD and NASA increasingly used computers and software to perform critical system functions, concern about the safety aspects of these components began to emerge. In the 1980s, DoD initiated efforts to integrate software into SSPs with the development of an extensive set of software safety tasks (300 series tasks) for incorporation into MIL-STD-882B (Notice 1).

The identification of separate software safety tasks in MIL-STD-882B focused engineering attention on the hazard risks associated with the software components of a system and critical effects on safety. However, the engineering community perceived these as segregated tasks to the overall system safety process and delegated the responsibility for performing these tasks to software engineers. This was an ineffective and inefficient process for handling software safety requirements because software engineers had little understanding of the system safety process and overall system safety functional requirements. Therefore, the separate software safety tasks were not included in MIL-STD-882C as separate tasks, but were integrated into the overall system-significant safety tasks. In addition, software engineers were given a clear responsibility and a defined role in the SSS process.

MIL-STD-882 defines system safety as “the application of engineering and management principles, criteria, and techniques to optimize all aspects of safety within the constraints of operational effectiveness, time, and cost throughout all phases of the system lifecycle.”

SSP objectives can be defined:

- Safety, consistent with mission requirements, is designed into the system in a timely, cost-effective manner
- Hazards associated with systems, subsystems, or equipment are identified, documented, tracked, evaluated, and eliminated or their associated risk is reduced to a level acceptable to the managing authority (MA) by evidence analysis throughout the entire lifecycle of a system
- Historical safety data, including lessons learned from other systems, are considered
- Minimum risk consistent with user needs is sought in accepting and using new design technology, materials, production, tests, and techniques; operational procedures must also be considered
- Actions taken to eliminate hazards or reduce risk to a level acceptable to the MA are documented
- Retrofit actions required to improve safety are minimized through the timely inclusion of safety design features during research, technology development, and acquisition of a system
- Changes in design, configuration, or mission requirements are accomplished in a manner that maintains a risk level acceptable to the MA
- Early consideration is given to safety, ease of disposal (including explosive ordnance disposal), and demilitarization of any hazardous materials (HM) associated with the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

system. Actions should be taken to minimize the use of HM, and therefore minimize the risks and lifecycle costs associated with HM use

- Significant safety data are documented as lessons learned and are submitted to data banks or as proposed changes to applicable design handbooks and specifications
- Safety is maintained and ensured after the incorporation and verification of engineering change proposals (ECPs) and other system-related changes.

With these definitions and objectives in mind, the system safety manager or engineer is the primary individual(s) responsible for the identification, tracking, elimination, and control of hazards or failure modes that exist in the design, development, test, and production of both hardware and software. This includes interfaces with the user, maintainer, and operational environment. System safety engineering is a proven and credible function supporting the design and systems engineering processes. The steps for managing, planning, analyzing, and coordinating system safety requirements are well established, and when implemented, successfully meet the above stated objectives.

System safety program requirements include:

- Eliminate identified hazards or reduce associated risk through design, including material selection or substitution
- Isolate hazardous substances, components, and operations from other activities, areas, personnel, and incompatible materials
- Locate equipment so that access during operations, servicing, maintenance, repair, or adjustment minimizes personnel exposure to hazards
- Minimize risk resulting from excessive environmental conditions (e.g., temperature, pressure, noise, toxicity, acceleration, and vibration)
- Design to minimize risk created by human error in the operation and support of the system
- Consider alternate approaches to minimize risk from hazards that cannot be eliminated. Such approaches include interlocks; redundancy; fail-safe design; fire suppression; and protective clothing, equipment, devices, and procedures
- Protect power sources, controls, and critical components of redundant subsystems by separation or shielding
- Ensure personnel and equipment protection (when alternate design approaches cannot eliminate the hazard) provide warning and caution notes in assembly, operations, maintenance, and repair instructions as well as distinctive markings on hazardous components and materials, equipment, and facilities. These shall be standardized in accordance with MA requirements
- Minimize severity of personnel injury or damage to equipment in the event of a mishap
- Design software-controlled or monitored functions to minimize initiation of hazardous events or mishaps

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Review design criteria for inadequate or overly restrictive safety requirements. If needed, recommend new design criteria supported by study, analyses, or test data.

An example of the need for, and the credibility of, a system safety engineering program is the Air Force aircraft mishap rate improvement since the establishment of the SSP in design, test, operations, support, and training processes. In the mid-1950s, the aircraft mishap rates were over 10 per 100,000 flight hours. Today, this rate has been reduced to about 1.25 per 100,000 flight hours.

Further information regarding the management and implementation of system safety engineering (and the analyses performed to support the goals and objectives of an SSP) is available through numerous technical resources. It is not the intent of this Handbook to be a comprehensive technical source for the subject of system safety, but to address the implementation of SSS within the discipline of system safety engineering. If specific system safety methods, techniques, or concepts remain unclear, please refer to the list of references in Appendix B for supplemental resources relating to the subject matter.

With the above information regarding system safety engineering (as a discipline) as a basis of understanding, a brief discussion must be presented as it applies to hazards and failure mode identification, categorization of safety risk in terms of probability and severity, and the methods of resolution. This concept must be understood to evolve to the accomplishment of software safety tasks within the system safety engineering discipline as defined in this Handbook.

3.6 Safety Risk Management

The process of system safety management and engineering is deceptively straightforward,[12] although it entails a great deal of work and linearity in a non-linear development and test process. The overall process is aimed at identifying system hazards and failure modes, determining causes, assessing mishap severity and probability of occurrence, determining hazard control requirements, verifying implementation, and identifying and quantifying any residual risk remaining prior to system deployment. This was reactive safety engineering. Today, using lessons-learned and systems engineering, experienced professional safety engineers can identify safety risk in the concept of operations (CONOPS) phase and identify and tag safety requirements before a system is designed. Where models and simulations are used, safety trade studies can be made to develop or fine-tune the required safety requirements.

Safety risk management focuses on the safety aspects of technical risk as it pertains to the conceptual system proposed for development. Safety risk management identifies and prioritizes hazards that are most severe or have the greatest probability of occurrence. The safety engineering process then identifies and implements safety risk elimination or reduction

12 *System Safety Analysis Handbook, A Resource Book For Safety Practitioners.* New Mexico Chapter of the International System Safety Society; July 1999.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

requirements for the design, development, test, and system activation phases of the development lifecycle.

As the concept of safety risk management is further defined, keep in mind that the value added is a reduction in the later effort needed to control safety risk for a program if the process is followed.

3.6.1 Initial Safety Risk Assessment

The efforts of the system safety engineer are launched by the performance of the initial safety risk assessment of the system. In the case of most DoD procurements, this takes place with the development of the Preliminary Hazard List (PHL) and the Preliminary Hazard Analysis (PHA). These analyses are discussed in detail later in this Handbook. This section of the Handbook will focus on the basic principles of system safety and hazard resolution. Specific discussions regarding how software influences or is related to hazards will be discussed in detail in Section 4.

3.6.1.1 Mishap, Hazard, and Failure Mode Identification

Safety analyses can be accomplished from numerous perspectives or levels of detail. The analyses can be accomplished from a mishap, hazard, or failure mode perspective. This detail is best visualized and understood when considering the simple fault tree diagram in Figure 3.3. As an example, an aircraft crash during takeoff would be an analysis performed at the mishap level, whereas loss of engine thrust would be a hazard that could lead to the mishap and could possibly be analyzed at that level. In addition, we may find that loss of engine compressor may be a required analysis at the failure mode level.

To determine the level at which the analysis is accomplished, a basic understanding of hazards and mishaps is required. A hazard can exist in a system, but that alone does not mean that anyone will be hurt or equipment will be damaged. When an existing hazard plus an initiating action occur, a mishap will result. Hazard analysis is accomplished to identify hazards that exist and the initiating action that will result in a mishap if it occurs. The risk assessment matrix (RAM) shows how serious the resulting mishap will be if it occurs. Regardless of where analysis begins, it is imperative to understand that all analysis performed must be “rolled up” to the mishap level for continuity, context, and mishap risk determination and acceptance. Just as hazard risk is determined by the hazard severity and the assigning of probabilities at the “cut set” events of the fault tree, mishap risk is determined in the same manner.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

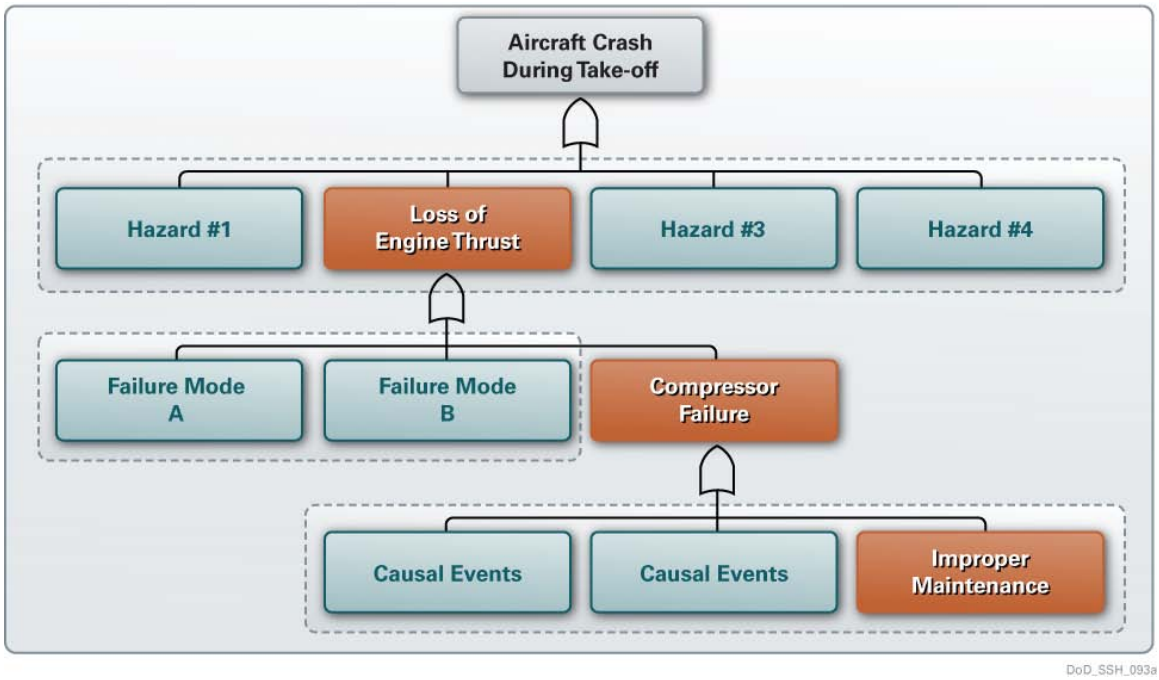

Figure 3-3: Mishap Risk Example

Traditionally, most analysis is performed at the hazard level. A hazard can be defined as a condition that is prerequisite to a mishap. The system safety engineer identifies the potential mishaps and the hazards, failure modes, and causal factors that could cause them. The initial hazard analysis and the Failure Modes and Effects Analysis (FMEA) accomplished by the reliability engineer provide the safety information required to perform the initial safety risk assessment of identified hazards. Without identified hazards and failure modes, very little can be accomplished to improve the overall safety of a system. Identified hazards, failure modes, and lessons learned become the basis for the identification and implementation of safety requirements within the design of the system. Once the mishaps and hazards are identified, they must be categorized in terms of safety risk (i.e., severity and probability).

3.6.1.2 Severity Categories

The first step in classifying safety risk requires the establishment of potential mishap severity for each hazard within the context of the system and user environments. This classification process uses the severity of damage and applies the number of times that the damage might occur. This process can also classify the potential for personnel injury or damage to the environment. Table 3-1 provides an example of how severity can be qualified.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table 3-1: Severity Categories

SEVERITY CATEGORIES		
Severity Category	Severity Level	Environment, Safety, and Occupational Health Mishap Result Criteria
Catastrophic	1	Could result in one or more of the following: death, permanent total disability, irreversible significant environmental impact, or loss exceeding $10M.
Critical	2	Could result in one or more of the following: permanent partial disability, injuries or occupational illness that may result in hospitalization of at least three personnel, reversible significant environmental impact, or loss exceeding $1M but less than $10M.
Marginal	3	Could result in one or more of the following: injury or occupational illness resulting in 10 or more lost work days, reversible moderate environmental impact, or loss exceeding $100K but less than $1M.
Negligible	4	Could result in one or more of the following: injury or illness resulting in less than 10 lost work days, minimal environmental impact, or loss less than $100K.

DoD_SSH_094

Note that this example is in the MIL-STD-882-specified format. The severity of mishap effect is qualitative and can be modified to meet the needs of a program. In order to assess safety severity, a benchmark against which to measure risk is essential. The benchmark allows for the establishment of a qualitative baseline that can be communicated across programmatic and technical interfaces. The baseline must be in a format that makes sense among individuals and between program interfaces. In today's SoS environment, there may need to be an integrated risk benchmark agreed to by all stakeholders and across each interface. SoS programs must schedule and budget for this added planning and coordination.

3.6.1.3 Probability Levels

The second half of the equation for determining safety risk is the identification of the probability of occurrence. The probability that a hazard could lead to the mishap without hazard controls can be determined by numerous statistical techniques. Statistical probabilities are usually obtained from reliability analyses pertaining to hardware component failures acquired through qualification programs. Component failure rates from reliability engineering are not always obtainable. This is especially true for advanced technology programs where component qualification programs do not exist and "one-of-a-kind" items are procured. Thus, the quantification of probability to a desired confidence level is not always possible for a specific mishap scenario. When this occurs, alternative analysis techniques are required for the qualification or quantification of mishap probability of hardware related nodes. Examples of credible alternatives include sensitivity analysis, event tree diagrams, and fault tree analysis (FTA). An example of the categorization of probability is provided in Table 3-2 and is in the format recommended by MIL-STD-882.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table 3-2: Probability Levels

PROBABILITY LEVELS			
Description	**Level**	**Specific Individual Item**	**Fleet or Inventory**
Frequent	A	Likely to occur often in the life of an item, with a probability of occurrence greater than 10^{-1} in that life.	Continuously experienced.
Probable	B	Will occur several times in the life of an item, with a probability of occurrence less than 10^{-1} but greater than 10^{-2} in that life.	Will occur frequently.
Occasional	C	Likely to occur sometime in the life of an item, with a probability of occurrence less than 10^{-2} but greater than 10^{-3} in that life.	Will occur several times.
Remote	D	Unlikely, but possible to occur in the life of an item, with a probability of occurrence less than 10^{-3} but greater than 10^{-6} in that life.	Unlikely, but can reasonably be expected to occur.
Improbable	E	So unlikely, it can be assumed occurrence may not be experienced in the life of an item, with a probability of occurrence of less than 10^{-6} in that life.	Unlikely to occur, but possible.
Eliminated	F	Incapable of occurrence in the life of an item; this category is used when potential hazards are identified and later eliminated.	Incapable of occurrence within the life of an item; this category is used when potential hazards are identified and later eliminated.

DoD_SSH_095b

3.6.1.4 Mishap Risk Index

As with the example provided for severity, Table 3-2 can be modified to meet the specification requirements of the user and developer. A systems engineering team (including system safety engineering) may choose to shift the probability numbers an order of magnitude in either direction or reduce the number of categories. All of the options are acceptable if the entire team is in agreement. This agreement must include the customer's opinions and specification requirements. The inclusion of individual units, entire populations, and time exposure intervals (periods) must also be considered when developing probability categories.

When integrated into a table format, mishap or hazard severity and probability produce the risk assessment matrix and the initial categorization for hazards prior to control requirements. The RAM of MIL-STD-882D Revision 1 is provided in Table 3-4. This matrix is divided into four levels of risk, as indicated by the color scale legend of the matrix. Red cells of the matrix are considered High risk. These risks require resolution or acceptance from the Acquisition Executive (AE). Orange cells are considered to be Serious risk while yellow cells are considered Medium risk. Green cells of the matrix represent Low risk. Those hazards deemed High or

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Serious should be redesigned or controlled to bring the risk to a level of acceptability. Those hazards deemed Medium or Low should be further assessed for options to minimize their risk where risk minimization is considered feasible.

Table 3-3: Risk Assessment Matrix

RISK ASSESSMENT MATRIX				
SEVERITY / PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)	High	High	Serious	Medium
Probable (B)	High	High	Serious	Medium
Occasional (C)	High	Serious	Medium	Low
Remote (D)	Serious	Medium	Medium	Low
Improbable (E)	Medium	Medium	Medium	Low
Eliminated (F)	Eliminated			

DoD_SSH_096

The primary benefit of the RAM is the ability and flexibility to prioritize hazards in terms of severity and probability. This prioritization of hazards allows the Program Manager, safety manager, and engineering manager to prioritize the expenditure and allocation of critical resources. A hazard with a Risk Assessment Code (RAC) of 3D should have fewer resources expended on safety analysis, design, test, and verification than a hazard with a RAC of 1B. Without the availability of the RAM, the allocation of resources becomes more arbitrary and potentially less effective.

Another benefit of the matrix is the accountability and responsibility of program and technical management to the system safety effort. The SwSSP identifies and assigns specific levels of management authority with the appropriate levels of safety mishap severity and probability. The RAC methodology holds program management and technical engineering accountable for the safety risk of the system during design, test, and operation and the residual risk upon delivery to the customer.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

From the perspective of the safety analyst, the RAM is a tool that is used during the entire system safety effort throughout the product lifecycle. Determination of the RAC is more complex when applied to the evaluation of system hazards and failure modes influenced by software inputs or software information. Alternatives to the RAC are discussed in detail in Section 4.

3.6.2 Safety Order of Precedence

The ability to adequately eliminate or control safety risk is predicated on the ability to accomplish the necessary tasks early in the design phases of the acquisition lifecycle. For example, it is more cost effective and technologically efficient to eliminate a known hazard by changing the design, rather than retrofitting a fleet in operational use. Because of this, the system safety engineering methodology employs a safety order of precedence for hazard elimination or control. When incorporated, the design order of precedence further eliminates or reduces the hazard's potential to culminate in mishap or failure mode initiation and propagate throughout the system. The following is extracted from MIL-STD-882.

- Design for Minimum Risk – From the first, design to eliminate hazards. If an identified hazard cannot be eliminated, reduce the associated risk to an acceptable level, as defined by the MA, through design selection.
- Incorporate Safety Devices – If identified hazards cannot be eliminated or their associated risk adequately reduced through design selection, that risk shall be reduced to a level acceptable to the MA through the use of fixed, automatic, or other protective safety design features or devices. Provisions shall be made for periodic functional checks of safety devices when applicable.
- Provide Warning Devices – When neither design nor safety devices can effectively eliminate identified hazards or adequately reduce associated risk, devices shall be used to detect the condition and produce an adequate warning signal to alert personnel of the hazard. Warning signals and their application shall be designed to minimize the probability of incorrect personnel reaction to the signals and shall be standardized within like types of systems.
- Develop Procedures and Training – Where it is impractical to eliminate hazards through design selection or adequately reduce the associated risk with safety and warning devices, procedures and training shall be used. However, without a specific waiver from the MA, no warning, caution, or other form of written advisory shall be used as the only risk reduction method for Category I or II Mishaps. Procedures may include the use of personal protective equipment. Precautionary notations shall be standardized as specified by the MA. Tasks and activities judged to be safety-critical by the MA may require certification of personnel proficiency.

3.6.3 Elimination or Risk Reduction

The process of hazard and failure mode elimination or risk reduction is based on the design order of precedence. Once hazards and failure modes are identified by evidence analysis and are

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

categorized, then specific (or functionally-derived) safety requirements must be identified for incorporation into the design for the elimination or control of safety risk. Defined requirements can be applicable to any of the four categories of the defined order of safety precedence. For example, a specific hazard may have several design requirements identified for incorporation into the system design. However, to further minimize the safety risk of the hazard, supplemental requirements may be appropriate for safety devices, warning devices, and operator and maintainer procedures and training. Most hazards have more than one design or risk reduction requirement unless the hazard is completely eliminated through the first (and only) design requirement. Figure 3-4 shows the process required to eliminate or control safety risk via the order of precedence described in Section 3.6.2.

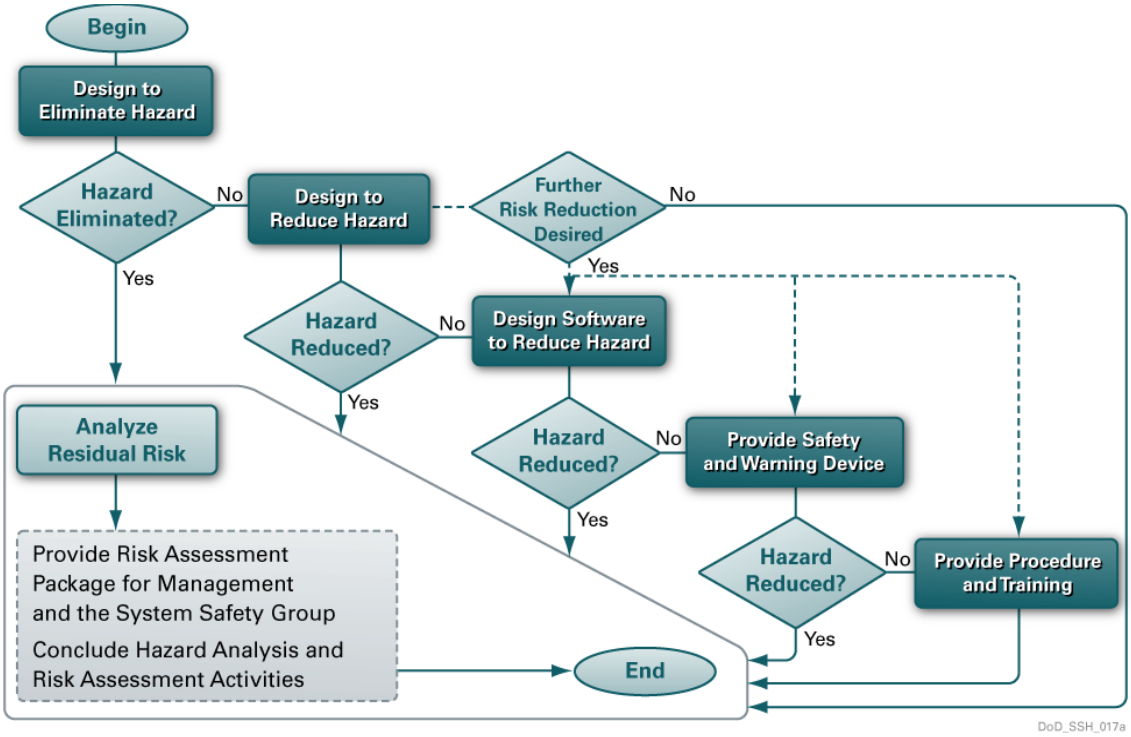

Figure 3-4: Hazard Reduction Order of Precedence

Identification of safety-specific requirements for the design and implementation portions of the system does not complete the safety task. The safety engineer must verify that the initial and derived requirements have been implemented as intended. Once hazard elimination and control requirements are identified and communicated to the appropriate design engineers, testing requirements must be identified for hazards which have been categorized as safety-significant. The safety risk must be communicated to the program's risk management group so that funding for the risk mitigation can be maintained. Cuts to funding, changes in schedule, and degradation in performance risks are also calculated and tracked at the PM level. Where software safety risk exists, the program risk cannot be closed until the software is validated in an operationally similar environment or test.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The categorization of safety risk in accordance with severity and probability must play a significant role in the depth of testing and requirements verification methods employed. Very low risk hazards do not require the same rigor of safety testing to verify the incorporation of requirements as those associated with safety-critical hazards. Where testing cannot always be accomplished, verification methods may be appropriate (e.g., designer sign-off on hazard record, as-built drawing review, and inspection of manufactured components).

3.6.4 Quantification of Residual Safety Risk

After the requirements are appropriately verified in the design and implemented (to the extent possible), the safety engineer must analyze each identified and documented hazard record to assess and analyze the residual risk within the system during operation and support activities. This is the same risk assessment process that was performed in the initial analysis described in Section 3.6.1. The primary difference in the analysis is the amount of design and test data available to support risk reduction activities.

After the incorporation of safety hazard elimination or reduction requirements, the hazard is once again assessed for potential mishap risk using mishap severity, mishap probability of occurrence, and the resulting RAC. A hazard with an initial mishap risk assessment of RAC 1B may have been reduced in safety risk to an RAC of 1D. However, since the hazard was not completely eliminated, there is a residual safety risk. This mishap risk is not as severe or as probable as the original, but the hazard and potential for mishap still exist.

The initial RAC of a hazard is determined during the PHA development prior to incorporation or implementation of requirements to control or reduce the safety risk, and is often an initial engineering judgment. The final RAC categorizes the hazard after the requirements have been implemented and verified by the developer. If hazards are not reduced sufficiently to meet the safety objectives and goals of the program, they must be reintroduced to safety engineering for further analyses and safety risk reduction. Risk is generally reduced within a probability category. Risk reduction across severity levels may require a hardware design change.

In conjunction with the safety analysis, engineering data, and information available, residual safety risk of the system, subsystems, user, maintainer, and tester interfaces must be quantified. Hazard records with remaining residual risk must be correlated within subsystems, interfaces, and the total system for the purpose of calculating the remaining risk. This risk must be communicated in detail (via the System Safety Working Group (SSWG) and the detailed hazard record system) to the Program Manager, the lead design engineers, the test manager, and the user and must be fully documented in the hazard database record. If residual risk is unacceptable to the Program Manager or risk acceptance authority, further direction and resources must be provided to the engineering effort.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

3.6.5 Managing and Assuming Residual Safety Risk

Managing safety risk can take a good amount of time, effort, and resources to accomplish. Referring to Table 3-3, specific categories must be established in the matrix to identify the level of management accountability, responsibility, and risk acceptance. Using Table 3-3 as an example, hazards with a RAC of 1A, 1B, 1C, 2A, and 2B are considered high risk. These hazards, if not reduced to a level below a RAC of 1C or 2B, cannot be officially closed without the Acquisition Executive's signature. This forces the accountability of assuming this particular risk to the appropriate level of management. The Program Manager can officially close hazards that are defined as low risk by the RAC.

Tables 3-1 through 3-3 have been extracted from MIL-STD-882D Revision 1. The tables provide a graphical representation of how a program may be set up with four levels of program and technical management. It is ideal to have the Program Manager as the official sign-off for all residual safety risk to maintain safety accountability. The PM is responsible for the safety of a product or system at the time of test and deployment. The safety manager must establish an accountability system for the assumption of residual safety risk based on user inputs, contractual obligations, and negotiations with the Program Manager. High and Serious safety risks in DoD programs are required to be briefed at milestone decisions. Residual risks must be accepted at high levels of management within Department of Defense management or the DoD Service.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4 Software System Safety Engineering

4.1 Introduction

This chapter of the Handbook introduces the managerial process and the technical methods and techniques inherent in the performance of software system safety tasks within a systems safety engineering and software development program. The chapter includes detailed tasks and techniques for the performance of safety analysis and for the traceability of software safety requirements from design to test. This chapter also provides the current best practices for establishing a credible and cost-effective software system safety program. An overview of this chapter is provided in Figure 4-1.

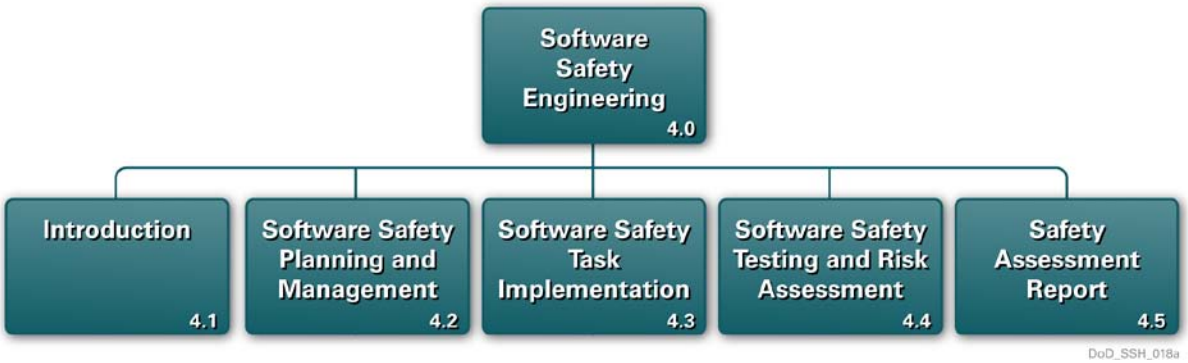

Figure 4-1: Chapter 4 Contents

A goal of this chapter is to formally identify the software safety duties and responsibilities assigned to the safety engineer, the software safety engineer, the software engineer, and the managerial and technical interfaces of each through sound systems engineering methods (Figure 4-2). This chapter identifies and focuses on the logical and practical relationships between the safety, design, and software disciplines. This chapter also provides the reader with the information necessary to assign software safety responsibilities and identify tasks attributed to system safety, software development, and hardware and digital systems engineering.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

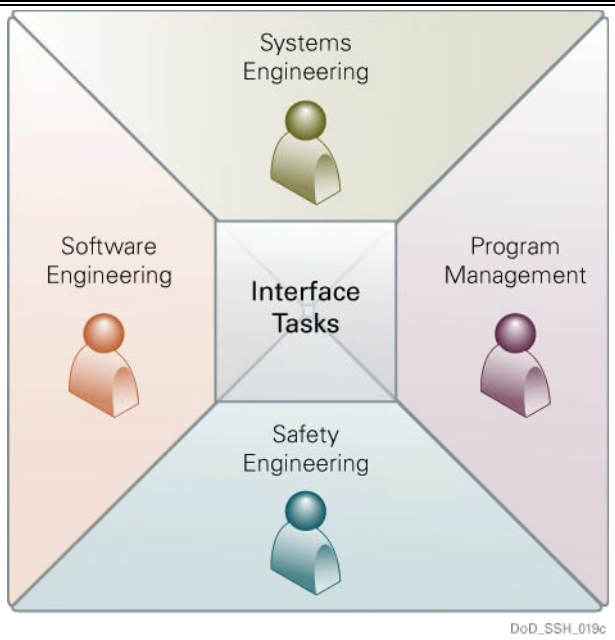

Figure 4-2: Software System Safety Interfaces

Section 4 is applicable to all managerial and technical disciplines involved in the development of safety-significant software. This section describes the processes, tools, and techniques used to reduce the safety risk of software operating in safety-critical systems. The primary purposes of this section are to:

- Define a recommended software safety engineering process that can be extracted, tailored, and implemented as program requirements for software system safety
- Define a recommended software assurance and integrity process
- Describe essential tasks to be accomplished by each professional discipline assigned to the software safety or software assurance and integrity tasks of the SSS team
- Identify interface relationships between professional disciplines and the individual tasks assigned to the SSS team
- Identify best practices to complete the software safety and software assurance and integrity processes
- Recommend tailoring actions to identify specific user requirements.

The accomplishment of a software safety management and engineering program requires careful forethought, adequate support from other disciplines, and timely application of expertise across the entire software lifecycle. Strict attention to planning is required to integrate the developer's resources, expertise, and experience with tasks to support contractual obligations or certification criteria established by the customer. Note that failing to perform a part of this process increases programmatic risk and must be documented and communicated in the programmatic risk area of the program. The depth and quality of the implemented tasks described in this section of the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Handbook reduces that programmatic risk. For example, the failure to perform planning will increase the probability that a safety failure in testing will occur and that the program will have to stop-fix-retest that individual function.

This section focuses on the establishment of a software safety program within the system safety engineering and software development processes. This section establishes a baseline program that, when properly implemented, will ensure that both initial software safety requirements and requirements specifically derived from functional hazards analysis are identified, prioritized, categorized, and traced through design, code, test, and acceptance. This section also ensures that specific levels of rigor (LOR) are identified, documented, and implemented for safety-critical and safety-related functions within the software development and test lifecycle.

This Handbook assumes a novice understanding of software safety engineering within the context of system safety and software engineering. Many topics of discussion within this section are considered constructs within basic system safety engineering. It is impossible to discuss software safety without including system safety engineering and management, systems engineering, software development, and program management.

4.1.1 Section 4 Format

This section is formatted to present both graphical and textual descriptions of the managerial and technical tasks that must be performed for a successful software safety engineering program. Each managerial process task, technical task, method, or technique will be formatted to provide:

- Graphical representation of the process step or technical method
- Introductory and supporting text
- Prerequisite requirements for task initiation
- Activities required to perform the task (including interdisciplinary interfaces)
- Associated subordinate tasks
- Critical task interfaces
- A description of the required task output(s) and product(s).

Additional information is located in Appendices A through G of this Handbook. The appendices provide practitioners with supplemental information and credible examples for guidance purposes. The titles of the appendices are:

- Appendix A – Definition of Terms
- Appendix B – References
- Appendix C – Handbook Supplemental Information
- Appendix D – COTS and Non-Developmental Item (NDI) Software
- Appendix E – Generic Software Safety Requirements (GSSR) and Guidelines

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Appendix F – Lessons Learned
- Appendix G – Example Request for Proposal (RFP) and Statement of Work

4.1.2 Process Charts

Each software safety engineering task possesses a supporting process chart. Each chart was developed to provide the engineer with a detailed "road map" for performing software safety engineering within the context of software design, code, and test activities. Figure 4-3 provides an example of the format for information considered for each process task. The depth of information presented in these figures includes inputs, outputs, primary sub-tasks, and critical interfaces. These process charts were trimmed from process worksheets to contain the information deemed essential for the effective management and implementation of the software safety program under the parent system safety program.

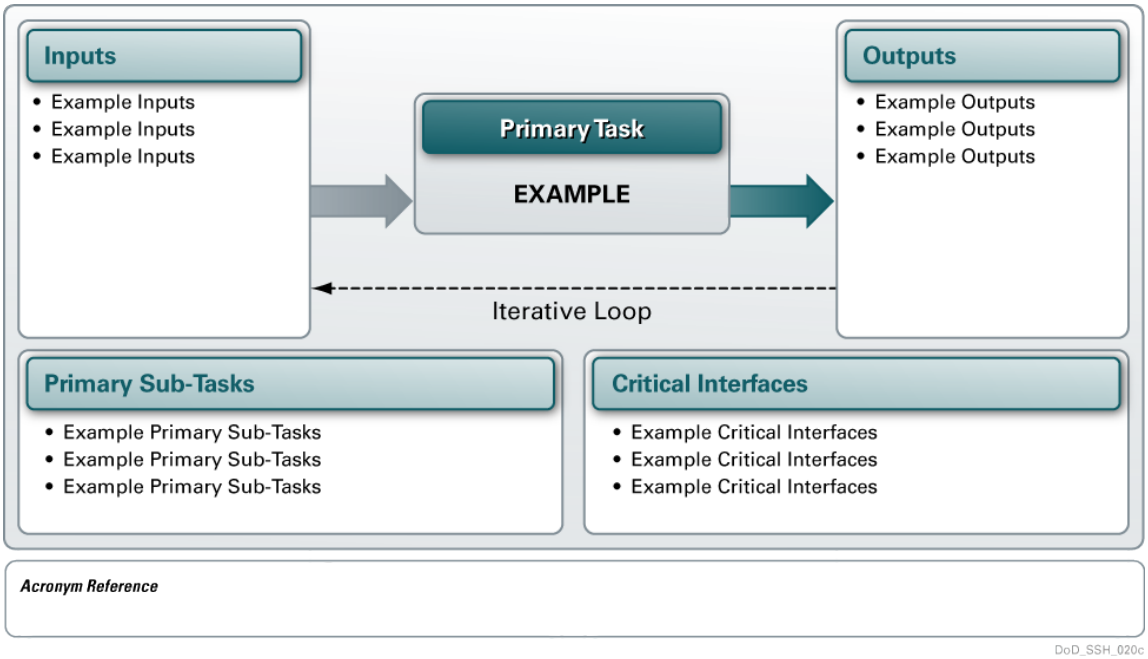

Figure 4-3: Process Chart Format Example

Each process chart presented in this handbook will contain:

- Primary task
- Task inputs
- Task outputs
- Primary sub-tasks

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Critical interfaces
- Acronym references

4.1.3 Software Safety Engineering Products

The specific products to be produced by the accomplishment of the software safety engineering tasks are difficult to segregate from those developed within the context of the SSP, the SEP, and the Software Development Plan (SDP). It is likely within individual programs that supplemental software safety documents and products will be produced to support the system safety effort. These may include supplemental analysis, data flow diagrams (DFDs), functional flow analysis, software requirement specifications, and Software Analysis Folders (SAF). This Handbook will identify and describe the documents and products that the software safety tasks will influence or generate. Specific documents include, but are not limited to:

- System Safety Program Plan or System Safety Management Plan (SSMP)
- Software Safety Program Plan (SwSPP)
- Generic Software Safety Requirements List
- Functional Hazard Analysis (FHA)
- Safety-Critical Functions List (SCFL)
- Preliminary Hazard List
- Preliminary Hazard Analysis
- Subsystem Hazard Analysis (SSHA)
- Safety Requirements Analysis (SRA)
- System Hazard Analysis
- Operation and Support Hazard Analysis (O&SHA)
- Safety Assessment Report (SAR)
- Safety Case.

It is important to note that each of the products listed, if contractually obligated, should be accomplished in accordance with the SOW, contract, data item description (DID), or approval/certification authority. Each individual contract may define the format, content, or approval criteria for these deliverables in slightly different ways to meet their specific needs. In addition, depending on the size of the program or project, some of these analyses may be a hardware/software combined analysis rather than separate documents.

4.2 Software Safety Planning and Management

Software safety planning and management is essential to a successful program (Figure 4-4). The software safety program must be integrated with and parallel to the SSP, the SDP, and program milestones. The software safety analyses must provide the necessary input to software

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

development milestones such that safety design requirements, implementation requirements, or design changes can be incorporated into the software with minimal impact. Program planning precedes all other phases of the SSS program and is perhaps the single most important step in the overall safety program. Inadequately specified safety requirements in the contractual documents may lead to program schedule and cost impacts when safety issues arise after the program has been initiated. These issues may require the acquirer to make residual risk acceptance decisions that can impact program safety, cost, schedule, and the safety of the end product lifecycle. The late establishment of safety program requirements and the late performance of the necessary safety analyses are likely to result in schedule delays, cost increases, and a potential increase in safety risk. Depending on how late in the program the process is implemented, potential safety risks associated with unassessed software must be determined and effectively communicated to management.

Figure 4-4: Software Safety Planning

Key topics to address during software safety planning and management definition include:

- Identification of the software assurance process as it relates to developing safe software using a Software Control Category (SCC), Software Criticality Level (SCI), and LOR (Level of Rigor) approach to software development and testing
- Identification of the software safety (hazard analysis) process as it relates to developing safe software by eliminating, mitigating, or controlling software-significant contributions to mishaps and hazards
- Identification of managerial and technical program interfaces required by the SSS team
- Definition of user and developer contractual relationships to ensure that the SSS team implements the tasks and produces the products required to meet contractual requirements
- Identification of programmatic and technical meetings and reviews supported by the SSS team

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Identification and allocation of critical resources to establish an SSS team and conduct a software safety program
- Definition of planning requirements for the execution of an effective program.

Figures 4-5 and 4-6 depict the primary differences between agencies that must be understood before considering the software safety planning and coordinating activities.

4.2.1 Planning

Comprehensive planning for the software safety program requires an initial assessment of the degree of software involvement in the system design and associated hazards. These efforts are challenging because little may be known about the system other than operational or functional requirements during the early planning stages. Part of the planning should include the review of previous and similar systems to identify lessons learned. In addition, it is imperative that all safety-significant terms and definitions are documented in the SOW to reduce conflict later in the lifecycle. A list of common terms and their definitions is included as Appendix A.

Figure 4-5 represents the basic inputs, outputs, tasks, and critical interfaces associated with Procuring Agency (PA) planning requirements. When system safety programs fail, it is often because of lack of adequate planning, including scope, task definition, and requirements. This can usually be traced back to the customer not ensuring that the RFP, SOW, and contract contain the correct language, terminology, and tasks to implement a safety program and provide the necessary resources. In part, this can be due to the WBS “hiding” discussed in Section 3. Software and system safety tasks integrated with systems engineering and integration or testing lines, combined with a poor SOW, contribute to inadequate funding by the stakeholder to accomplish the required safety tasks. The ultimate success of any safety program strongly depends on the planning function by the customer.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

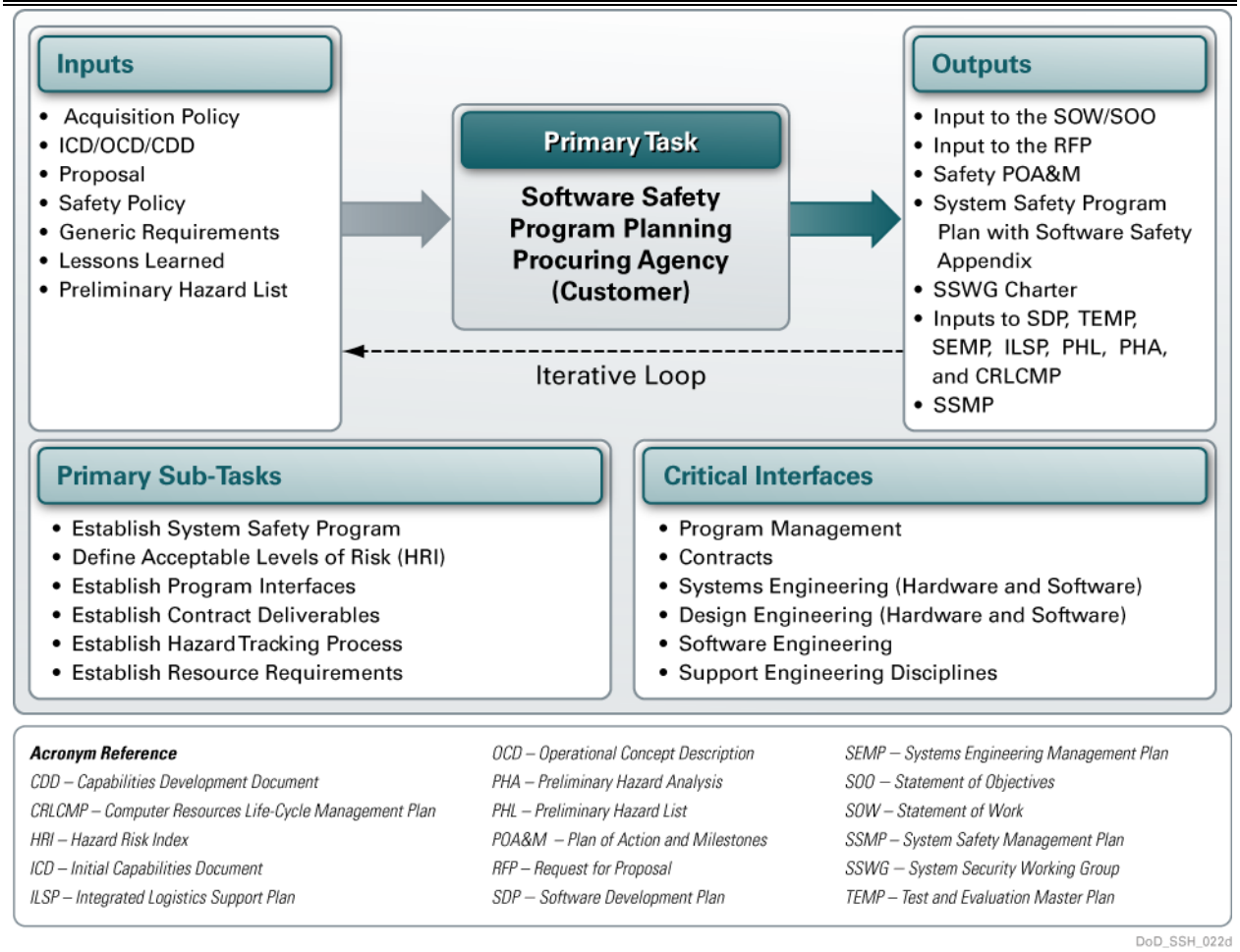

Figure 4-5: Software Safety Planning by the PA

For the PA, software safety program planning begins as soon as the need for the system is identified. The PA must identify points of contact (POCs) within the organization and define the interfaces between various engineering disciplines, administrative support organizations, program management, contracting groups, and Integrated Product Teams (IPTs) to develop the necessary requirements and specifications documents. The PA must incorporate the necessary language into any contractual documents to ensure that the system under development will meet the safety acceptance or certification criterion.

PA safety program planning continues through contract award and may require periodic update during initial system development and as the development proceeds through various phases. Management of the overall System Safety Program continues through system delivery, acceptance, and throughout the system's lifecycle. After deployment, the PA must continue to track system hazards and risks and monitor the system in the field for safety concerns identified by the user. The PA must also make provisions for safety program planning and management

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

for any system upgrades, product improvements, maintenance, technology refreshment, and other follow-on efforts.

The major milestones affecting the PA's safety and software safety program planning include the release of contract requests for proposals or quotes, proposal evaluation, major program milestones, system acceptance and certification testing and evaluation, production contract award, initial operational capability (release to the users), and system upgrades or product improvements.

Although the Developing Agency's (DA's) software safety program planning begins after receipt of a contract RFP or quotes, the DA can significantly enhance its ability to establish an effective program through prior planning (see Figure 4-6). Prior planning includes establishing effective systems engineering and software engineering processes that fully integrate system and software systems safety. Corporate engineering standards and practices documents that incorporate the tenets of system safety provide a strong baseline from which to build a successful system safety program even though the contract may not contain specific language regarding the safety effort.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

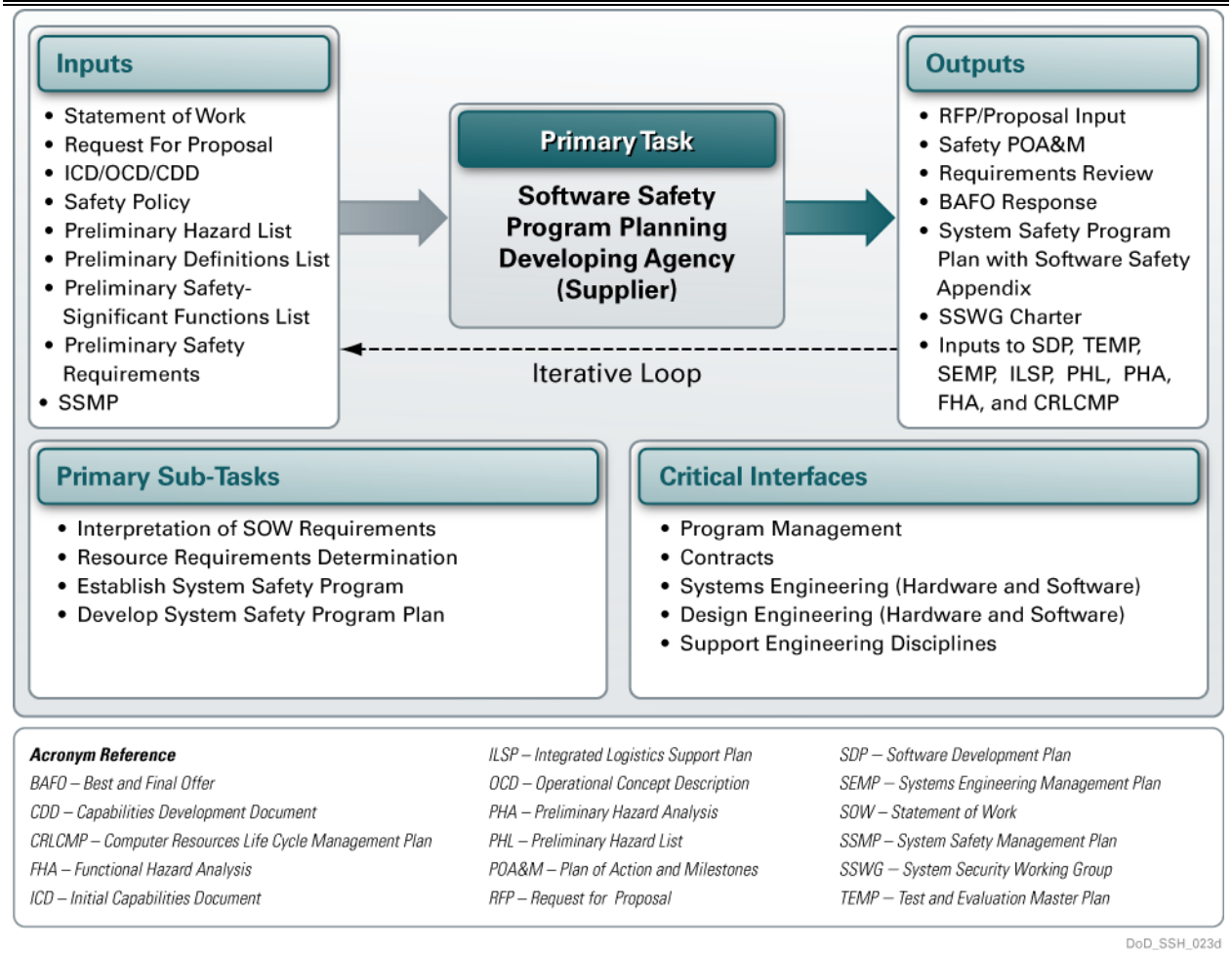

Figure 4-6: Software Safety Planning by the DA

Acquisition reform recommends that the Government take a more interactive approach to system development without interfering with that development. An example of this approach is to participate as a member of the DA's IPT as an advisor without hindering development. From the system safety and SSS perspective, active participation in the appropriate IPTs provides the Government perspective on recommendations and decisions made in those forums. Participation also requires the Government representative to alert the developer to known mishaps, hazards, and failure modes collected through lessons learned and other historical reviews. These items may not be readily available to the developer.

Contract language is often non-specific and does not provide detailed system safety requirements. Therefore, it is the DA's responsibility to define a comprehensive SSP that will ensure that the delivered system provides an acceptably low level of safety risk to the customer, not only for the customer's benefit, but for the DA's benefit as well. At the same time, the DA must remain competitive and reduce safety program costs to the lowest practical level consistent with ensuring the delivery of a system with the lowest practical risk. Although the preceding

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

discussion focused on the interaction between the Government and the DA, the same tenets apply to any contractual relationship, including between prime and subcontractors.

The DA software safety planning continues after contract award and requires periodic updates as the system proceeds through various phases of lifecycle development. These updates should be in concert with the PA's software safety plans. Management of the overall system and SSS programs continues from contract award through system delivery and acceptance and may extend throughout the system lifecycle, depending on the type of contract. If the contract requires the DA to perform routine maintenance, technology refreshments, or system upgrades, software safety program management and engineering must continue throughout the system's lifecycle. In this case, the DA must make provisions for safety program planning and management for these phases and other follow-on efforts for the system.

The major milestones affecting the DA's safety and software safety program planning include the receipt of contract requests for proposals or quotes, contract award, major program milestones, system acceptance testing and evaluation, production contract award, release to the customer, system upgrades, and product improvements.

While the software safety planning objectives of the PA and DA may be similar, the planning and coordination required to meet these objectives may come from different perspectives (in terms of specific tasks and their implementation), yet they must be in concert (Figure 4-7). Both agencies must work together to meet the safety objectives of the program. In terms of planning, this includes:

- Establishment of a system safety program (see Section 4.2.1.1)
- Definition of acceptable levels of safety risk (see Section 4.2.1.2)
- Development of software assurance process (see Section 4.2.1.3)
- Development of a Software Criticality Matrix (SCM) (see Section 4.2.1.4)
- Development of the level of rigor table (see Section 4.2.1.5)
- Definition of critical program, management, and engineering interfaces (see Section 4.2.1.6)
- Definition of contract deliverables (See Section 4.2.1.7)
- Definition of safety-significant terms (see Appendix A)

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

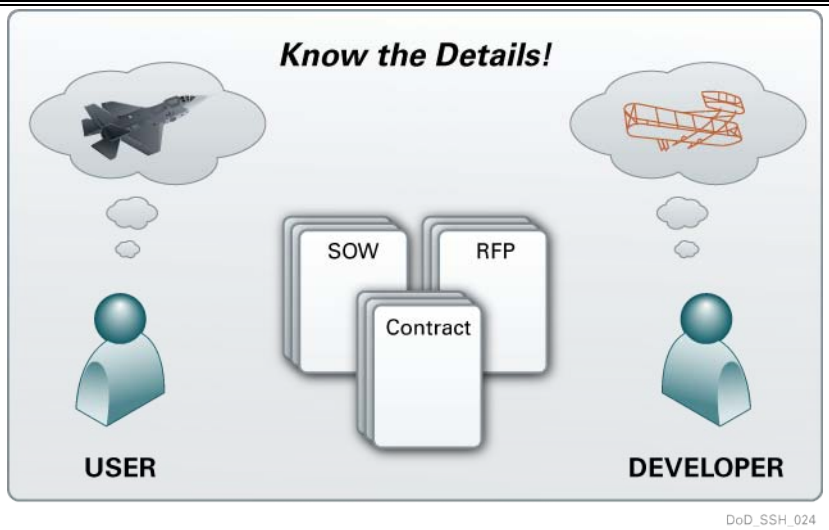

Figure 4-7: Planning the Safety Criteria is Important

4.2.1.1 Establish the System Safety Program

The PA must establish the safety program as early as practical in the development of the system. The Program Manager should identify a Principal for Safety (PFS) or other safety manager early in the program to serve as the single POC for all safety-significant matters on the system. This individual will interface with safety review authorities, the DA safety team, PA and DA program management, the safety engineering team, the software development and test teams, and other technical disciplines to ensure that the safety program is effective and efficient. The PFS may also establish and chair a Software Systems Safety Working Group (SwSSWG) or SSS team. For large system developments where software is likely to be a major portion of the development, a safety engineer for software may also be identified who reports directly to the overall system PFS. The size of the safety organization will depend on the complexity of the system under development and the inherent safety risks. The size of the PM's safety team is also influenced by the degree of interaction with the customer, supplier, and other engineering and program disciplines. If the development approach is a team effort with a high degree of interaction between the organizations, the safety organization may require additional personnel to provide adequate support.

The PA should prepare a System Safety Management Plan describing the overall safety effort within the PA organization and the interface between the PA's safety organization and the DA's system safety organization. The SSMP is similar to the SSPP in that it describes the roles and responsibilities of the program office individuals with respect to the overall safety effort. The PFS or safety manager should coordinate the SSMP with the developing agency's SSPP to ensure that the tasks and responsibilities are complete and will provide the desired risk assessment. The SSMP differs from the SSPP in that it does not describe the details of the safety program contained in the SSPP, such as analysis tasks. Programs initiated under MIL-STD-882D are not required to have an SSPP. However, Section 4.2 of this Handbook requires that the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Program Manager and the developer document the agreed upon system safety process. This is virtually identical to the role of the SSPP. The PFS or safety manager must coordinate the SSMP with this documented safety process.

The PA must specify the software safety program for programs where software performs or influences safety-critical functions of the system. The PA must establish the team in accordance with contractual requirements, managerial and technical interfaces and agreements, and the results of all planning activities discussed in previous paragraphs of this Handbook. Proper and detailed planning will increase the probability of program success. The tasks and activities associated with the establishment of the SSP are applicable to both the supplier and the customer.

The degree of influence of the software on safety-critical functions in the system is often not known until the design progresses to the point of functional allocation of requirements at the system level. The Program Manager must predicate the software safety program on the goals and objectives of the system safety and software development disciplines of the proposed program. The safety program must focus on the identification and tracking (from design, code, and test) of both initial SSRs and guidelines and those requirements derived from system-specific functional hazards analyses. A sound SSS program traces both sets of requirements through test and requirements verification activities. The ability to identify all applicable SSRs is essential and must be adequately addressed.

4.2.1.2 Defining Acceptable Levels of Risk

One of the key elements in safety program planning is the identification of the acceptable level of risk for the system. This process requires both the identification of a RAC and a statement of the goals of the safety program for the system. The former establishes a standardized means with which to group hazards by risk (e.g., unacceptable or undesirable), while the latter provides a statement of the expected safety quality of the system. The ability to categorize specific hazards into the RAC matrix is based on the ability of the safety engineer to assess the mishap severity and likelihood of occurrence (see Section 3.6.1.4). The PA, in concert with the user, must develop a definition of the acceptable risk and provide that to the DA. The PA must also provide the developer with guidance on risk acceptance authorities and reporting requirements. DoDI 5000.02 requires that High risk hazards (unacceptable hazards per MIL-STD-882) obtain Component Acquisition Executive (CAE) signature for acceptance. Serious risk hazards (undesirable) require acceptance at the Program Executive Officer (PEO) level. The DA must provide the Program Manager with supporting documentation for the risk acceptance authority.

4.2.1.3 Planning for Two Distinct Processes

The successful execution of a total software safety program consists of two distinct separate but overlapping processes that can individually result in the development of safer software. When these two processes are implemented and integrated together, as illustrated in Figure 4-8, the product of the two processes can produce software as safe as reasonably practical. This integrated approach uses the strengths and skills of each individual SSS team member to carry out specific tasks within their individual domain expertise.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

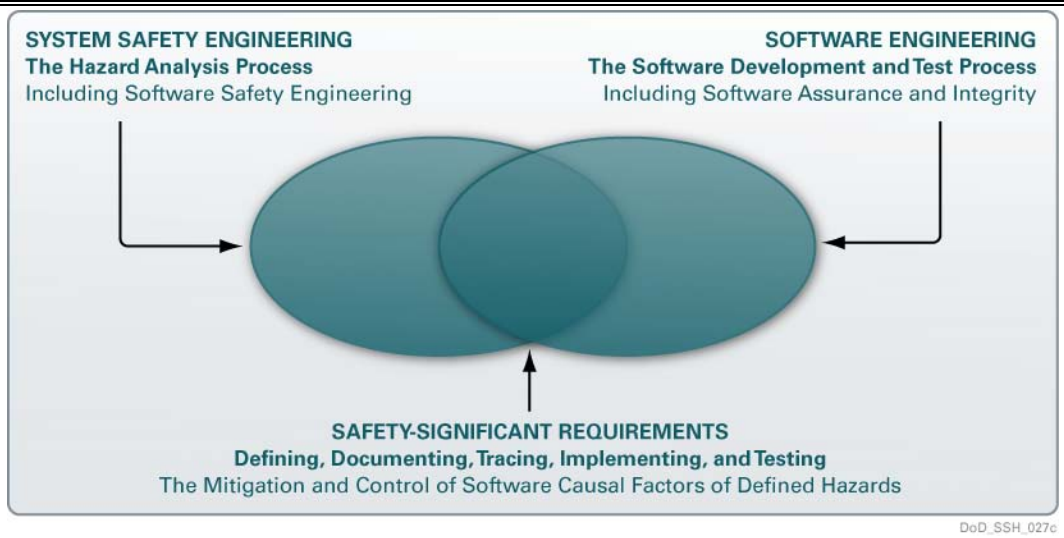

Figure 4-8: Two Distinct Processes of Software Safety Engineering

Individual Government agencies, Military Services, and commercial contractors must ensure that both processes are fully addressed in their planning documents, including the definition of the process itself; the tasks required to implement the process; the artifacts that the tasks will produce; and evidences for customer approval, acceptance, or certification. Specific planning documents combine elements of these two processes and include the SSPP, SSMP, SDP, Software Test Plan (STP), Configuration Management Plan, and Software Quality Assurance (SQA) Plan.

Both the software hazard analysis process and the software assurance process are requirements-based processes. Safety-specific software development, test, and quality assurance (QA) process implementation requirements bring assurance and integrity to safety-significant functions. Generic and functionally-derived mishap and hazard mitigation requirements bring success to the software safety hazard analysis process.

4.2.1.3.1 Software Safety Assurance and Integrity Process

The software safety assurance and integrity process is based on the identification, categorization, implementation, and verification of safety-significant functionality of a system and the robustness (or LOR) used within the process to increase the confidence or assurance that:

- The safety-significant functions are positively identified in hardware, software, and firmware domains
- The safety-significant functions are mapped to the architecture, interfaces, and designs

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- The defined safety-significant function(s) accomplishes what it was defined to accomplish, when it was supposed to be accomplished, and that it performed in the correct time or sequence of defined operations
- The defined safety-significant functions do not possess any functional capability that they were not defined to possess
- The defined safety-significant functions possess the appropriate integrity within the design (as defined by the SCI level of rigor tasks) for fault detection, isolation, annunciation, tolerance, and recovery.

This process is initially based on the defined safety-significant functions of the system and the safety severity of the consequences of loss of function, malfunction, degraded function, or functioning out of time or sequence. Once the safety severity has been identified for the function, the function is then assessed against the software control categories of MIL-STD-882 (or other Design Assurance Level (DAL)-related definitions from governing standards or certification criteria) to assess its level of autonomy, command and control authority, or safety information display attributes within the context of the system. This allows the software safety team to identify the Safety Integrity Level (SIL) definitions for the assessment of an appropriate level-of-rigor assignment within the software development and test processes.

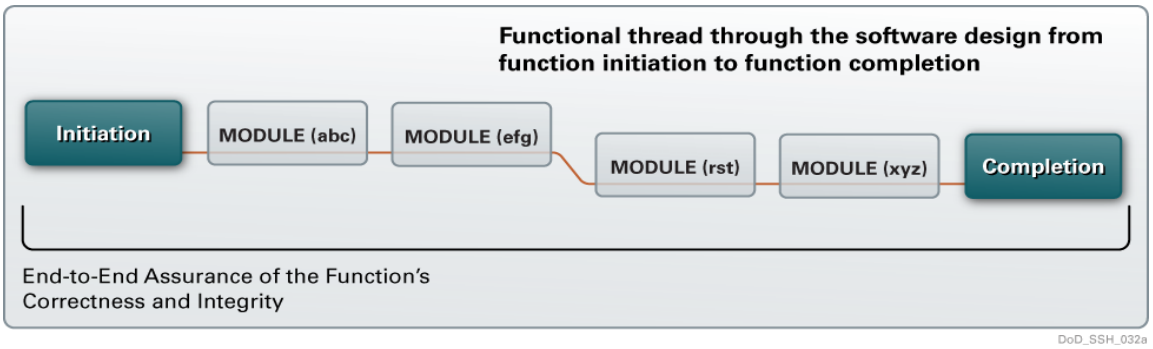

Figure 4-9: Graphical Depiction of Software Assurance and Integrity

Safety-significant functions are identified using functional analysis techniques, including the FHA (as defined by standards such as Society of Automotive Engineers Aerospace Recommended Practice (SAE ARP) 4761). The severity of loss of function or the function failing in any manner is assessed from a "credible/normal" worst case perspective. While mishap severity and command and control authority form the basis for LOR assigned in the software development and test processes, it is their successful implementation that helps reduce the probability that the defined mishaps or hazards will occur in the nominal and off-nominal operations of the system. The necessary SIL levels and LOR tables will be further discussed in Sections 4.2.1.4 and 4.2.1.5.

Figure 4-9 provides an overview of the development of the product of a software assurance and integrity process as an end-to-end assurance of an individual safety-significant function in the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

design architecture. The engineering artifacts developed over the lifecycle of the product prove the functional assurance and integrity of the process is based on process, review, inspection, verification, validation, and test requirements defined by the level-of-rigor assignment for that given function.

4.2.1.3.2 Software Safety Hazard Analysis Process

The software safety hazard analysis process identifies the potential mishaps and hazards of the system and the specific contributions of the software to cause, influence, contribute to, or mitigate the failure modes of the mishap or hazard occurrence. While the software safety assurance and integrity process verifies the end-to-end attributes of the safety-significant functions within the design architecture, the software can still contribute to mishaps or hazards based on the individual pathways to failure. Hazard analysis identifies the specific failure mode pathways in the context of hardware, software, and human error contributions. This analysis identifies the specific points of failure for the safety-significant software functionality of the system.

Figure 4-10 illustrates that a given safety-significant function may possess assurance and integrity in terms of end-to-end continuity, but can still initiate or contribute to a hazard occurrence in the context of failure for a specific hazard. There can be several points of failure within a specific function as it applies to hazards of the system. These specific points of failure are identified and resolved through the hazard analysis process. In Figure 4-10, the hardware sensor interface with the software is the initiation causal factor for this specific failure mode pathway. The software safety analysis process must:

- Identify the failure
- Define the mechanisms to reduce the likelihood of occurrence
- Define the requirements for failure detection, annunciation, tolerance, or recovery.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

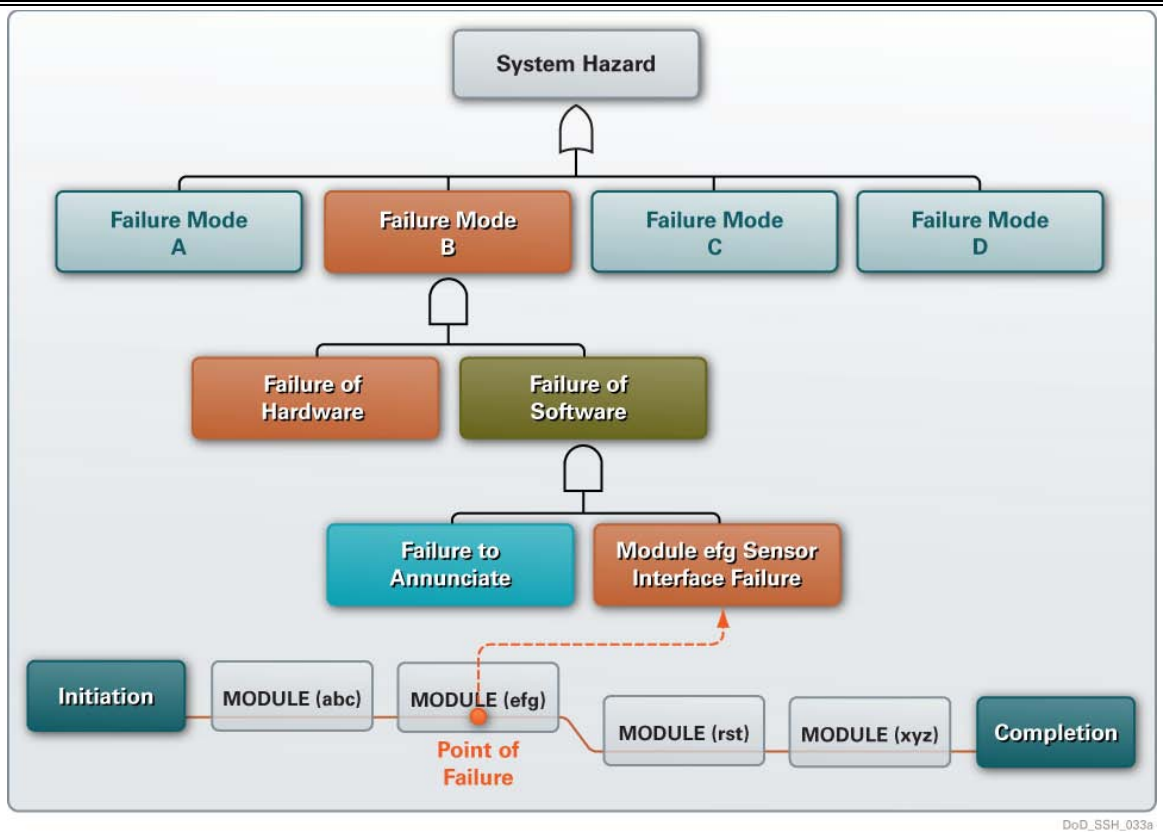

Figure 4-10: Graphical Depiction of Software Safety Hazard Analysis

4.2.1.4 Defining and Using the Software Criticality Matrix

There are several SIL-type software assurance approaches, along with their individual specific SIL definitions. Two of the most used definitions are presented in MIL-STD-882C and RTCA DO-178B (see Figure 4-11). These approaches do not determine software-caused hazard probabilities, but they assess the severity of the system's safety-significant functions and the software's control capability in context of the software's ability to implement the functions. In doing so, each of the software safety-significant functions can be labeled with a software control category for the purpose of defining the level of rigor that will be required in the function's design, implementation, test, and verification. The SSS team must review these lists and tailor them to meet the objectives of the SSP and software development program.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

MIL-STD-882C	RTCA-DO-178B
(I) Software exercises autonomous control over potentially hazardous hardware systems, subsystems, or components without the possibility of intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazards occurrence.	**(A)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.
(IIa) Software exercises control over potentially hazardous hardware systems, subsystems, or components, allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate.	**(B)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a hazardous/severe/major failure condition of the aircraft.
(IIb) Software item displays information requiring immediate operator action to mitigate a hazard. Software failure will allow or fail to prevent the hazard's occurrence.	**(C)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a major failure condition for the aircraft.
(IIIa) Software item issues commands over potentially hazardous hardware systems, subsystems, or components, requiring human action to complete the control function. There are several redundant, independent safety measures for each hazardous event.	**(D)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.
(IIIb) Software generates information of a safety critical nature used to make safety-critical decisions. There are several redundant, independent safety measures for each hazardous event.	**(E)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of function with no effect on aircraft operational capability or pilot workload. Once software has been confirmed as level E by the certification authority, no further guidelines of this document apply.
(IV) Software does not control safety-critical hardware systems, subsystems, or components and does not provide safety-critical information.	

DoD_SSH_030b

Figure 4-11: Legacy Software Control Category Definitions

Table 4-1 introduces the application of new software control category definitions for safety-significant programs that include unmanned systems (UMSs) and system-of-systems. These definitions are a blending of historical definitions with inputs from legacy systems and lessons learned that are appropriate for modern programs.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table 4-1: Software Control Category Definitions

SOFTWARE CONTROL CATEGORIES (SCC)		
Level	**Name**	**Description**
1	Autonomous (AT)	• Software functionality that exercises autonomous control authority over potentially safety significant hardware systems, subsystems, or components without the possibility of predetermined safe detection and intervention by a control entity to preclude the occurrence of a mishap or hazard. *(This definition includes complex system/software functionality with multiple subsystems, interacting parallel processors, multiple interfaces, and safety-critical functions that are time critical)*
2	Semi-Autonomous (SAT)	• Software functionality that exercises control authority over potentially safety significant hardware systems, subsystems, and components, allowing time for predetermined safe detection and intervention by independent safety mechanisms to mitigate or control the mishap or hazard. *(This definition includes the control of moderately complex system/software functionality, no parallel processing, or few interfaces, but other safety systems/mechanisms can partially mitigate. System and software fault detection and annunciation notifies the control entity of the need for required safety actions)* • Software item that displays safety significant information requiring immediate operator entity to execute a predetermined action for mitigation or control over a mishap or hazard. Software exception, failure, fault, or delay will allow, or fail to prevent, mishap occurrence. *(This definition assumes that the safety-critical display information may be time-critical, but the time available does not exceed the time required for adequate control entity response and hazard control)*
3	Redundant Fault Tolerant (RFT)	• Software functionality that issues commands over safety-significant hardware systems, subsystems, or components requiring a control entity to complete the command function. The system detection and functional reaction includes redundant, independent fault tolerant mechanisms for each defined hazardous condition. *(This definition assumes that there is adequate fault detection, annunciation, tolerance, and system recovery to prevent the hazard occurrence if software fails, malfunctions, or degrades. There are redundant sources of safety-critical or safety significant information, and mitigating functionality can respond within any time-critical period)* • Software that generates information of a safety-critical nature used to make critical decisions. The system includes several redundant, independent fault tolerant mechanisms for each hazardous condition, detection and display.
4	Influential	• Software generates information of a safety-related nature used to make decisions by the operator but does not require operator action to avoid a mishap.
5	No Safety Impact (NSI)	• Software functionality that does not possess command or control authority over safety significant hardware systems, subsystems, or components and does not provide safety significant information. Software does not provide safety significant or time sensitive data or information that requires control entity interaction. Software does not transport or resolve communication of safety significant or time-sensitive data.

DoD_MIL-STD_882_004

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The concept of labeling safety-significant functions within the software with control capabilities may be foreign to some software developers and programmers. This activity is necessary for the identification and prioritization of software entities that possess safety implications. It is through this prioritization that safety-significant code can receive the appropriate robustness and level of rigor over the lifecycle, while effectively managing the critical resources of the program. The most important aspect of the activity is that the software with the highest level of control over safety-significant hardware must receive more attention or level of rigor than software with less safety risk potential. Autonomous software with functional links to catastrophic hazards demands more coverage than software that influences low severity hazards. This methodology helps prioritize and manage the critical resources of schedule, budget, and personnel associated with the development of the system.

The Software Criticality Matrix assists Program Managers, the SSS team, and the subsystem and system software designers in allocating resources to the software safety effort. This is not a RAC matrix for software. The SCM is a mechanism to assess the software command, control, and autonomy authority for a specific safety-significant function and to determine the LOR required in the software development and test activities to ensure its safety integrity within the system context. The higher the Software Criticality Index number, the greater potential there is that fewer resources will be required to verify that the software will execute safely in the context of the system or SoS. The software control measure of the SCM also assists in the prioritization of software design and programming tasks. The SCI's greatest value is during the functional allocation phase. Using the SCI, a strong software safety program can:

- Reduce the autonomy of the software control of safety-critical aspects of the system
- Provide adequate functional (modules) and physical (processors) partitioning of safety-significant functions
- Allow the software designer to design the safety-significant functions to reduce the probability of non-essential or non-safety-significant functionality contributing to failure
- Minimize the number of safety-critical functions in the software
- Minimize the complexity of each safety-critical function
- Use the software to reduce the risk of other hazards in the system design.

If the analysis of the conceptual design (architecture) shows a high degree of autonomy over safety-critical functions, the software safety effort requires significantly more resources in the design, code, and test phases. Therefore, the systems engineering team should consider this factor early in the design phases. By reducing the number of software modules containing safety-critical functions, the developer reduces the portion of the software requiring safety assessment and assurance, and thus, the resources required for those tasks. The systems engineering team must balance these issues with the required and desired capabilities and complexities of the system. Developers often use software to control functionality when non-software alternatives will provide the same capabilities. Developers use this approach to save weight, reduce maintenance complexity, reduce power required, or reduce heat created by the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

alternative. While the safety risk associated with the non-software alternatives must still be assessed, the process is likely to be less costly and resource intensive.

Figure 4-12 illustrates that software functions must be defined as either "safety significant" or "not safety significant." This figure further illustrates that those functions that are safety significant must be defined as either safety critical or safety related, based on safety severity definitions. Be advised that individual Government contracts may not share these terms and definitions even though the safety community is working diligently to standardize terms and definitions across programs, Military Services, and Government agencies.

Figure 4-12: Recommended Terms and Definitions

Safety-significant functions that are identified by the FHA (or other means) are assessed for safety consequence if they should fail, malfunction, or function out of time or out of sequence. This safety consequence assessment must consider the "credible-normal" worst case safety consequence and the severity to determine whether it resides in the Catastrophic, Critical, Marginal, or Negligible categories. Once determined, the safety-significant function is then assessed to determine the autonomy, command and control authority, or the generation of safety-significant information within the system context (Table 4-2).

Safety-significant software definitions used on some programs of record narrow the definition of "safety-critical" to only those software functions which are single points of failure. The difficulty with these narrowed definitions is the size and complexity of the system where the single point of failure may not exist in a local system, but may exist in the larger system of systems. These narrow definitions are not recommended and are often rejected in Joint Services review boards because of the SoS lessons learned.

Failure to adequately define these definitions prior to the requirements and the design phase will result in incorrect decisions in systems engineering, software engineering, and test engineering. These incorrect decisions can further manifest themselves in poor safety requirements, functional partitioning, configuration control, and COTS selection criteria. Inadequate criticality definitions can also adversely affect the selection of network firmware and middleware.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table 4-2: Software Criticality Matrix

SCC \ Severity	Catastrophic	Critical	Marginal	Negligible
AT Autonomous	1	1	3	4
SAT Semi Autonomous	1	2	3	4
RFT Redundant Fault Tolerant	2	3	4	4
Influential	3	4	4	4
NSI No Safety Impact	5	5	5	5

DoD_SSH_074d

Once a safety-significant function has been defined and assessed for severity and software control category, the resultant SCI of 1 through 5 dictates the LOR requirements for that specific function. Remember, the Software Criticality Index is not the same as the mishap or hazard risk index, though they appear similar. A low index number from the software criticality matrix does not mean that a design is unacceptable. Rather, it indicates that a more significant level of effort is necessary for the requirements definition, design, implementation, and test of the software and its interactions with the system.

4.2.1.5 Defining the Requirements for Level of Rigor

The development of safe software is dependent on the definition of software safety requirements that are to be levied upon the software development and test processes, the design architecture, and any implementation methods or tools. The LOR table establishes the early requirements that must be implemented for the SCI assessment for individual safety-significant functions of the system. Figure 4-13 provides an example template of a LOR table that could be used on a program that possesses safety-significant functionality. These tables and their defined requirements should be tailored for the specific program. This tailoring must be agreed on by the customer and the supplier and must be adequately covered in the RFP, SOW, SSMP, SSPP, SDP, and STP.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

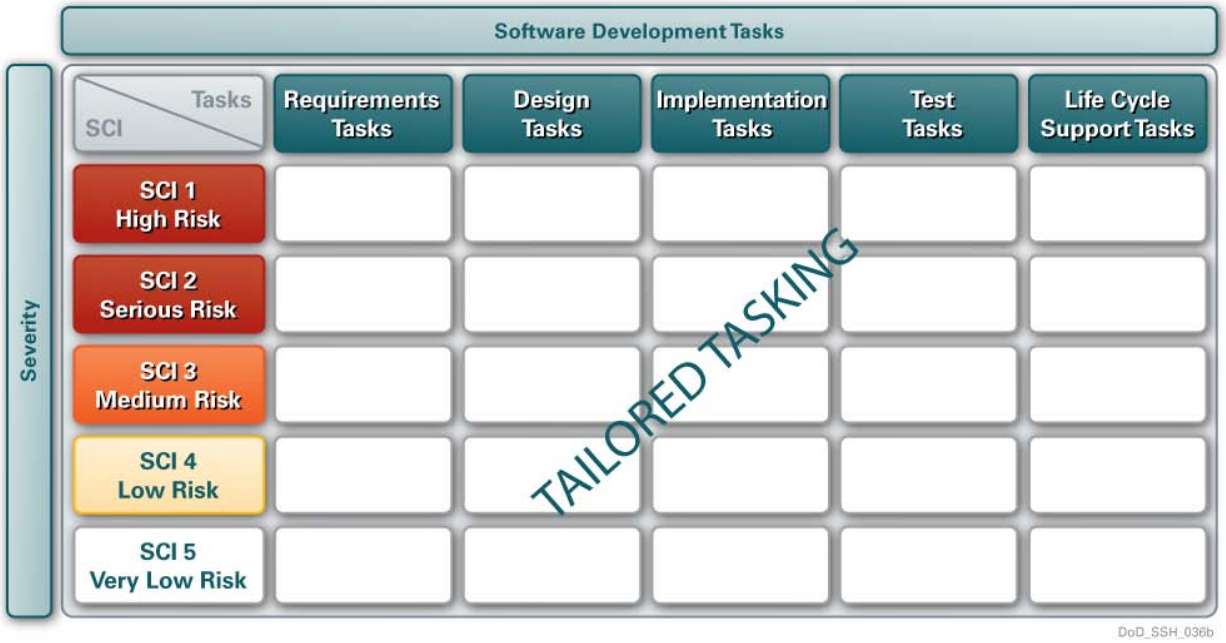

Figure 4-13: Example LOR Template

The individual program tailoring of the LOR table template is based on the acceptance or certification criteria of the customer. The table may have more or fewer columns and may be tied to software lifecycle phases, or the table could have fewer rows based on fewer granularities of the severity definitions. For example, a program may have only three SCI levels of rigor based on the needs or requirements of the program. In addition, there is an inherent risk associated with not accomplishing the tasks indicated in the LOR table. Each task identified on the LOR table, if successfully implemented, will help to reduce the likelihood of software contributing to a hazard or mishap occurrence. However, this reduced likelihood remains a qualitative engineering judgment and will be addressed in Paragraph 4.2.1.8.2, Mishap Probability. The risk that must be accepted should these tasks not be accomplished must be communicated to program management and the design team and should be included in the SAR.

The level of rigor task requirements that must be populated into the table are the product of customer and supplier tailoring. Table 4-3 provides a list of tasks and requirements that can be considered as the table is populated. This list is a product of lessons learned from other programs.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table 4-3: Example LOR Tasks or Requirements

Design Requirements	Process Tasks	Test Tasks
Fault Tolerant Design	Design Reviews	Safety-Significant Function Testing
Fault Detection	Safety Reviews	Functional Thread Testing
Fault Isolation	Design Walkthroughs	Limited Regression Testing
Fault Annunciation	Code Walkthroughs	100% Regression Testing
Fault Recovery	Independent Reviews	Failure Modes and Effects Testing
Warnings Cautions, and Advisories	Independent Walkthroughs	Safety-Critical Interface Testing
Redundancy	Traceability of Safety-Significant Requirements to Design	COTS, Government Off-the-Shelf Input, Output Test, and Verification
Independence	Traceability of Safety-Significant Requirements to Code	Independent Testing of Prioritized Safety-Related Functions
Functional Partitioning	Traceability of Safety-Significant Requirements to Test	Functional Qualification Testing
Physical Partitioning	Safety Test Results Review	Verification and Validation
Design Safety Standards	Software Quality Assurance Inspections and Audits	Independent Verification and Validation
Design Safety Guidelines	Traceability of Safety-Significant Requirements to Hazards	Full Screening of All COTS Features
Design Safety Lessons Learned	Specific Software Language Requirements	
Full COTS Features Disclosure and Analysis		

DoD_SSH_075c

The possible tasks and requirements identified in Table 4-3 are intentionally left vague to reduce the temptation to cut and paste the table into other project-specific documents. The intended purpose of this table is to provide a list of subjects that could be discussed and considered in the development of a LOR table for a specific program.

An example of a project-specific LOR table is provided in Table 4-4, where the customer and supplier agreed on a program of three levels of rigor. For each defined task, the task is followed (in brackets) by the individual or group responsible for the implementation of that specific task.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The oversight of the application of the LOR tables must be included in the systems engineering technical reviews and the milestone reviews. Of particular interest would be the amount of re-assessment, re-analysis, and re-testing performed for a given change. The amount and scope of regression testing should be driven by safety criticality (and the risk of not performing), and not by budget or schedule.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table 4-4: Example of Specific LOR Tasks

Level of Rigor	Requirements Tasks	Design Tasks	Implementation Tasks	Test and V&V Tasks	Lifecycle Support Tasks
LOR-1 High Risk	• All LOR-2 and -3 Tasks • Create requirements for a fault tolerant design [Safety and Requirements and Design] • Create requirements to ensure that all interfaces are validated and controlled at all times [Safety and Requirements and Design]	• All LOR-2 and -3 Tasks • Functionally partition all implementations of LOR-3 requirements from lower levels of rigor in the design [Design] • Update design to be stress tolerant [Design] • Update design for SCF (and only SCF) philosophy [Safety and Design] • Update design to control functional, physical, and human interfaces [Design]	• All LOR-2 and -3 Tasks • Create a unit test plan defining the criteria for unit testing of safety-critical code [Safety and Development] • Review unit test results and verify that the unit tests provide the required unit test coverage and were executed in compliance with the unit test plan [Safety and Test] • Perform detailed inspections of code for compliance with safety-critical coding standards and guidelines [Test] • Perform detailed code inspections for fault contributions [Safety, Test, and Development] • Create unit tests with goal of approaching 100% source code branch-point unit testing [Development]	• All LOR-2 and -3 Tasks • Add safety-critical integration test cases to the formal test plan(s) [Safety and someone independent of the developer] • Execute safety-critical integration and test cases [Someone independent of the developer] • Add fault injection safety-critical test cases to the formal test plan(s) [Safety and Test] • Execute fault injection testing [Test] • Add test cases to the Regression Test Plan to support 100% regression testing on all LOR-3 software [Safety, Test, and Development] • Perform 100% regression testing on all LOR-3 software that is changed [Test] • Test to verified and validated interfaces [Test] • Perform code walkthroughs and review all LOR-3 code for safety issues [Safety and Development] • Add safety test cases to verify that all functional, physical, and human interfaces are under continuous control [Safety and Test] • Execute complete decision coverage of the code [Test] • Execute complete modified condition/decision coverage of the code [Test]	• All LOR-2 and -3 Tasks
LOR-2 Med Risk	• All LOR-3 Tasks • Review safety-critical requirements for completeness [Safety]	• All LOR-3 Tasks • Perform a Sub-System Hazard Analysis [Safety] • Functionally partition all implementations of LOR-2 requirements from lower levels of rigor in the design [Design] • Incorporate fault isolation, annunciation, and tolerance into the design [Design]	• All LOR-3 Tasks • Perform high-level reviews of code for compliance with safety-critical coding standards and guidelines [Safety and Development] • Independently witness the execution of unit tests [Safety and Test] • Review unit test plan [Safety]	• All LOR-3 Tasks • Create test cases for safety-critical code to test for [Safety, Test, and Development]: - Stress testing - Stability testing - Disaster testing • Review each LOR-2+ test case [Safety] • Participate in test anomaly resolution [Safety] • Plan, perform, and review failure modes and effects testing (FMET) plans and procedures. • Plan, perform, and review functional and FMET regression test plans and procedures.	• All LOR-3 Tasks • Review defects for safety impact [CM and Safety] • Review and give signature approval on safety-critical Change Requests (CRs) [Safety] • Independently review and check in code changes to CM [someone other than the author of the changes]
LOR-3 Low Risk	• Perform a System Hazard Analysis [Safety] • Create a traceability matrix from safety-critical requirements (contributing or mitigating) to identified hazards (of initial RAC Medium or High) [Safety] • Review safety-critical requirements and prioritize for future builds [Safety and Requirements] • Create requirements to ensure that safety-critical interfaces are validated and controlled at all times [Safety, Requirements, and Design] • Map safety requirements to functions and into views of the system and software architectures, labeling COTS and NDI as they become "make-or-buy" outcomes.	• Continue System Hazard Analysis [Safety] • Follow design guidelines for safety-critical design [Design] • Analyze the design (including functional systems and software architectures and interfaces) for failure modes and hazard contributions [Safety and Design] • Review the design for compliance with the design guidelines and for safety issues [Safety and Design] • Review of the User Interface design for safety issues [Safety and Design] • Create traceability from design components to safety-critical requirements [Requirements and Design]	• Continue System Hazard Analysis [Safety] • Mark safety-critical code with the appropriate LOR [Development] • Follow coding guidelines and comply with coding standards for safety-critical code [Development] • Create traceability from code to safety-critical design requirements [Design and Development] • Execute unit tests [Development] • Participate in acceptance review of safety-critical code [Safety] • Create a safety-critical test report documenting the safety-critical unit testing compliance and execution results [Safety and Test]	• Continue System Hazard Analysis [Safety] • Create test cases for safety-critical code [Safety, Test, and Development] - Exception handling correctness - Fault handling correctness - Interface correctness - Boundary handling correctness • Review safety-critical test results and verify that the safety-critical test cases provide the required test coverage and were executed in compliance with the formal test plans [Safety and Test] • Create traceability between safety-critical test cases and safety-critical requirements [Safety and Test] • Mark safety-critical test cases with the appropriate LOR [Safety and Test] • Create a safety-critical test report documenting the safety-critical formal testing compliance and execution results [Safety and Test] • Calculate and document the residual safety risk (after mitigation) [Safety] • Review all traceability matrices for coverage and completeness [Safety and Design]	• Review proposed CRs for safety impact [Safety and Requirements] • Mark safety-critical items in CM with the appropriate LOR [Development and CM] • Document the results of any Safety Reviews [Safety] • Review problem reporting/defect tracking, change control, and change review activities for safety impact and compliance [CM and Safety] • Perform an Operations and Support Hazard Analysis [Safety]

4.2.1.6 Program Interfaces

System safety engineering is responsible for the coordination, initiation, and implementation of the software safety engineering program. While this responsibility cannot be delegated to any

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

other engineering discipline within the development team, software safety must assign specific tasks to the engineers with the appropriate domain expertise. Historically, system safety engineering identifies, assesses, and eliminates or reduces the safety risk of hazards associated with complex systems. As software becomes a major aspect of the system, software safety engineering must establish and perform the required tasks and establish the technical interfaces required to fulfill the goals and objectives of the system safety (and software safety) program. However, the SSS team cannot accomplish this requirement without intercommunication and support from other managerial and technical functions.

Within DoD acquisition and product development agencies, IPTs are the usual mechanism to ensure the success of the design, manufacture, fabrication, test, and deployment of systems. These IPTs formally establish the accountability and responsibility between functions and among team members. This accountability and responsibility is both from the top down (management-to-team member) and from the bottom up (team member-to-management).

The establishment of a credible SSS activity within an organization requires rigor in the identification of team members, the definition of program interfaces, and the establishment of lines of communication. Establishing formal and defined interfaces allows program and engineering managers to assign required expertise for the performance of the identified tasks of the software safety engineering process. Figure 4-14 shows the common interfaces necessary to adequately support a SwSSP. Common interfaces include management, technical, and contractual interfaces.

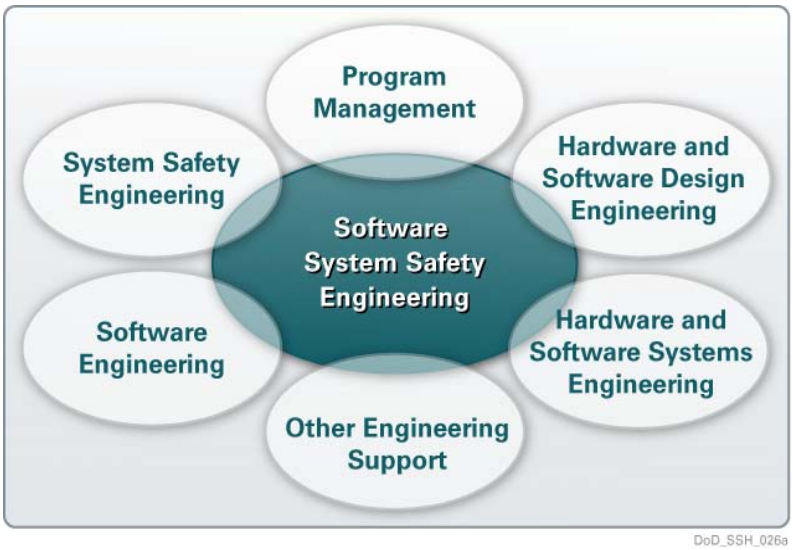

Figure 4-14: Software Safety Program Interfaces

"Other Engineering Support" identified in Figure 4-14 includes such disciplines as quality assurance, configuration management (CM), testing, and any other interfaces being used by the safety engineer.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.2.1.6.1 Management Interfaces

The Program Manager, under the authority of the AE or the PEO:

- Coordinates the activities of each professional discipline for the entire program
- Allocates program resources
- Approves the program planning documents, including the SSMP or SSPP
- Reviews safety analyses; accepts impacts on the system for Critical and higher category hazards (based on acceptable levels of risk); and submits findings to the PEO for acceptance of unmitigated, unacceptable hazards.

The Program Manager is responsible for ensuring that processes are in place that meet the programmatic, technical, and safety objectives, and also the functional and system specifications and acceptance or certification requirements of the customer. The PM must allocate critical resources within the program to reduce the sociopolitical, managerial, financial, technical, and safety risk of the product. Management support is essential to the success of the SSS program.

The Program Manager ensures that the safety team develops a practical process and implements the tasks required to:

- Perform a functional hazard assessment
- Identify safety-related and safety-critical functions
- Identify generic safety requirements for the system and software specifications
- Define and ensure the implementation of LOR tasks
- Identify system hazards
- Categorize hazards in terms of severity and likelihood
- Perform causal factor analysis
- Derive hardware and software design requirements to eliminate or control hazards
- Provide evidence for the implementation of generic and derived hardware and software safety design requirements
- Analyze and assess the residual safety risk of any hazards that remain in the design at the time of system deployment and operation
- Report the residual safety risk and hazards associated with the fielded system to the appropriate acceptance or certification authority.

The safety manager and the software engineering manager depend on program management for the allocation of necessary resources (time, tools, training, money, and personnel) to successfully complete required SSS engineering tasks.

Within the DoD framework, the AE/PEO is ultimately responsible for the acceptance of the residual safety risk at the time of test, initial systems operation, and deployment (note that PMs

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

can usually accept Low residual safety risk). The AE/PEO must certify at the Test Readiness Review (TRR) and the Safety Review Board (SRB) that all hazards and failure modes have been eliminated or that the risk is mitigated or controlled to a level as low as reasonably practical. At this critical time, an accurate assessment on the residual safety risk of a system facilitates informed management and engineering decisions.

Under legacy acquisition processes without the safety risk assessment provided by a credible system safety process, the AE/PEO will assume unreasonable personal, professional, programmatic, and political liabilities in the decision-making process. If the Program Manager failed to implement effective system and SSS programs, the risk acceptance authority may assume the liability due to failure to follow DoD directives. Under acquisition reform, the developer now assumes much of the liability, but at the cost of Government control. The developer is only liable for the implementation and test of the requirements. If the desired safety attributes of the system or SoS are not defined as requirements, they will not be present in the delivered system.

The ability of the PFS or safety manager to provide an accurate assessment of safety risk depends on the support provided by program management throughout the design and development of the system. Under acquisition reform, the Government purchases systems as if they are off-the-shelf products. The developer warrants the system for performance and safety characteristics against the requirements in the contract, making the developer liable for any mishaps that occur. However, under the U.S. law (Title 10), the AE/PEO is ultimately responsible for the safety of the system and the assessment and acceptance of the residual risk. The developer's safety team, in coordination with the PA's safety team, must provide the AE/PEO with the accurate assessment of the residual risk so that they can make informed decisions. Residual risk is the sum of the unmitigated hazards covered by the requirements plus all other hazards found during the analysis and test that were not covered by requirements.

4.2.1.6.2 Technical Interfaces

The engineering disciplines associated with system development must provide technical support to the SSS team (see Figure 4-15). Engineering management, design engineers, systems engineers, software development engineers, integrated logistics support, and other domain engineers supply this essential engineering support. Other domain engineers include reliability, human factors, quality assurance, test and evaluation, V&V, maintainability, survivability, and supportability. Each member of the engineering team must provide timely support to the defined processes of the SSS team to accomplish the safety analyses and specific design influence activities which eliminate, reduce, or control hazard risk. This includes the traceability of SSRs from the user/customer capabilities definition to design to test (and test results), with associated and documented evidence of implementation.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

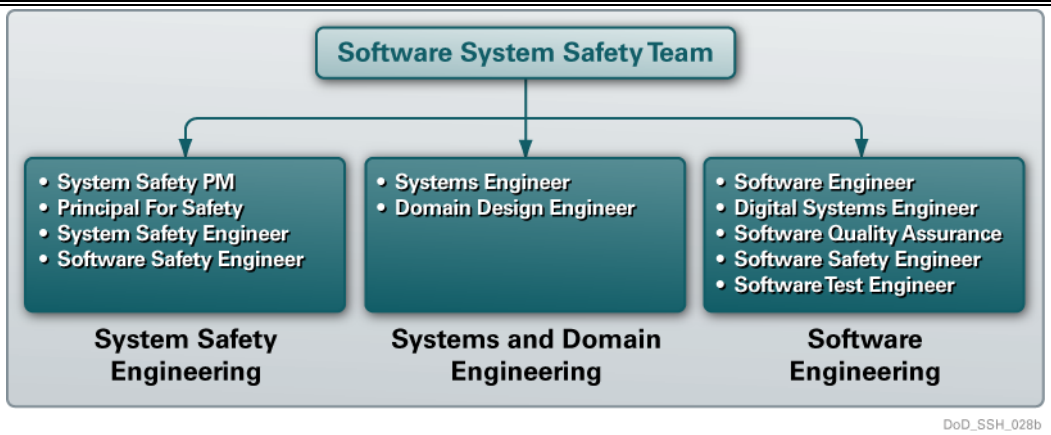

Figure 4-15: Proposed SSS Team Membership

The software safety activity will fail if software engineering acceptance and support of the software safety process, functions, and implementation tasks are not secured. Most formal education and training for software engineers and developers does not present, teach, or rationalize system safety. The system safety process relating to the derivation of functional SSR through hazard analyses is foreign to most software developers. In many cases, the concept that software can be a causal factor to a hazard was once a unique idea to many software engineers. In this world of software-controlled, safety-critical systems, this is no longer the case.

A successful SSS effort requires the establishment of a technical SSS team approach. The SSP manager, in concert with the systems engineer and software engineering team leaders, must define the individual tasks and specific team expertise required and assign responsibility and accountability for the accomplishment of LOR tasks. The SSPP must include the identification and definition of the required expertise and tasks in the software safety portion or appendix.

The team must identify both the generic SSRs and guidelines and the functional safety design requirements derived from system hazards and failure modes that have specific software input or influence. Once these hazards and failure modes are identified, the team can identify specific safety design requirements through an integrated effort. All SSRs must be traceable to test and must be correct, consistent, complete, and testable. The Requirements Traceability Matrix (RTM) tool used by the design team or within the SRA documents this traceability. The implemented requirements must eliminate, control, or reduce the safety risk as low as reasonably practical while meeting the user requirements and acceptance criteria within operational constraints. Appendix C.5 contains supplemental information pertaining to the technical interfaces.

4.2.1.6.3 Contractual Interfaces

Management planning for the SSS function includes the identification of contractual interfaces and obligations. Each program has the potential to present unique challenges to system safety,

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

software development, and test managers. These challenges may include an RFP that does not specifically address the safety of the system or contract deliverables that are costly to develop. Regardless of the challenges, the tasks needed to accomplish an SSS program must be planned to meet both the system and user specifications and requirements and to ensure the safety goals and acceptance or certification criteria of the program. Essential contractual obligations include:

- RFP
- SOW
- Contract
- Contract Deliverable Requirements List (CDRL)

Example templates of an RFP and SOW/Statement of Objectives (SOO) are located in Appendix G.

4.2.1.7 Contract Deliverables

The SOW defines the deliverable documents and products (CDRLs) required by the customer. Each CDRL should be addressed in the SSPP and should include the necessary activities and process steps required for production and approval. Completion of contract deliverables is normally tied to the acquisition lifecycle of the system being produced and the program milestones identified in the System Engineering Master Plan (SEMP). The planning required by the system safety manager ensures that the system safety and software safety processes provide the necessary data and output for the successful accomplishment of the plans and analysis. The system safety schedule should track closely to the SEMP and should be proactive and responsive to both the customer and the design team.

Contract deliverables should be addressed individually on the safety master schedule and within the SSPP. These documents are either contractual deliverables or internal documents required to support the development and test effort.

Current acquisition policy limits specific military and DoD standards and few system safety deliverables. The PA must ensure that sufficient deliverables are identified and contractually required to meet programmatic and technical objectives. Some programs have an Integrated Development Environment (IDE) where the Government has full access to the provider's development environment. This allows the Government access to data to perform its own checks, audits, analyses, simulations, and tests on program data. However, the IDE environment is not yet prevalent on all programs.

This activity must also specify the content, format, and acceptance criteria of each deliverable item. As existing Government standards transition to commercial standards and guidance, the safety manager must ensure that sufficient planning is accomplished to specify the breadth, depth, and timeline of each deliverable (normally defined by DIDs). The breadth and depth of the deliverable items must provide the necessary audit trail to ensure that safety levels of risk are

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

achieved (and are visible) during development, test, support transition, and maintenance in the out-years. The deliverables must also provide the necessary evidence or audit trail for validation and verification of SSRs. The primary method for maintaining a sufficient audit trail is the use of a developer's Safety Data Library (SDL). This library would be the repository for all safety documentation. Appendix C.3 describes the contractual requirements that should be contained in the system SDL.

Common deliverable supplements associated with a software safety and software assurance program that support the common deliverables of a system safety program include, but are not limited to:

- Defined terms and their definitions
- Functional hazard assessment and analysis
- Safety-critical and safety-related functions list
- Safety requirements criteria analysis
- Level of rigor table
- Software safety requirements traceability (from hazards to Software Requirements Specifications (SRS), and from SRS to design, code, and test)
- Code-level analysis.

4.2.1.8 Development of the Mishap Risk Index

Criteria described in MIL-STD-882 provide the basis for the RAC (described in Section 3.6.1.4). This example may be used for guidance, or an alternate RAC may be proposed. The given RAC methodology used by a program must possess the capability to graphically delineate the boundaries between High, Serious, Medium, and Low risk. The PA is responsible for defining and documenting the Risk Acceptance Matrix for the program. Figure 4-16 provides a graphical representation of an example safety risk acceptance matrix.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Example Risk Acceptance Matrix

Frequency of Occurrence	1 Catastrophic	2 Critical	3 Marginal	4 Negligible
A – Frequent	High	High	Serious	Medium
B – Probable	High	High	Serious	Medium
C – Occasional	High	Serious	Medium	Low
D – Remote	Serious	Medium	Medium	Low
E – Improbable	Medium	Medium	Medium	Low

DoD_SSH_025e

High Risk	**Unacceptable,** without Program Executive Officer concurrence, the condition must be resolved. Design action is required to eliminate or control hazard.
Serious Risk	**Undesirable,** Program Executive Officer decision is required. The hazard must be controlled or hazard probability reduced.
Medium Risk	**Allowable,** with Program Manager review. Hazard control is desirable if cost effective.
Low Risk	**Acceptable,** with Program Manager review. Normally, Low risk is not cost effective to control. The hazard is either negligible or can be assumed to not occur.

Figure 4-16: Example of Risk Acceptance Matrix

The ability to categorize specific hazards into the matrix is based on the ability of the safety engineer to assess the severity and likelihood of hazard occurrence. The traditional RAC matrix did not include the influence of the software on the hazard occurrence. The traditional RAC is based on the mishap severity and probability of occurrence to assign the RAC with probabilities defined in terms of mean time between failures, probability of failure per operation, exposure intervals, or probability of failure during the lifecycle depending on the nature of the system. This relies heavily on the ability to obtain component reliability information from engineering sources. However, applying probabilities of this nature to software, except in purely qualitative terms, is impractical. Therefore, obtaining confidence that software will not cause mishap or hazard occurrence is predicated on the implementation of a sound software assurance and integrity process that supplements the software safety hazard analysis process.

Software does not fail in the same manner as hardware. Software does not wear out, break, or have increasing tolerances that result in failures. Software errors are generally errors in the requirements (e.g., failure to anticipate a set of conditions that lead to a hazard, or the influence of an external component failure on the software) or implementation errors (e.g., coding errors,

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

incorrect interpretation of design requirements). If the conditions occur that contribute to the software not performing as expected, a failure occurs. Therefore, reliability predictions become a prediction of when the specific conditions will occur that cause it to fail. Without the ability to accurately predict a software error occurrence, hazard categorization can only be accomplished qualitatively through sound engineering judgment using causal analysis and software assurance processes.

When successfully specified and implemented, the software assurance and integrity process reduces the likelihood of software contributing to failure based on the level of rigor in the software development and test process. The hazard analysis process identifies specific points of failure within the design architecture for the purpose of defining specific hazard mitigating or control requirements. When these hazard mitigating requirements are successfully implemented and verified/validated in the design through inspection and test, it reduces the likelihood of the software contributing to a hazard occurrence.

During the early phases of the safety program, the prioritization and categorization of hazards is essential for the allocation of resources to the functional areas responsible for resolving the hazards. This section of the Handbook presents a method of categorizing hazards having software causal factors strictly for purposes of allocation of resources to the SSS program. This methodology does not provide an assessment of the residual risk associated with the software at the completion of development. However, the execution of the safety program, the development and analysis of SSRs, and the verification of their implementation in the final software provide the basis for a qualitative assessment of the residual risk in traditional terms.

4.2.1.8.1 Mishap Severity

Regardless of the hazard causal factors (hardware, software, human error, or environment), the severity of the hazard usually remains constant in most instances. The consequence of a hazard's occurrence is defined by considering the credible, normal worst case event regardless of what caused the hazard unless the design of the system somehow changes the possible consequence. Because the mishap severity remains constant, the severity table presented in Section 3.6.1.2 (Table 3-1, Severity Categories) remains an applicable criterion for the determination of safety criticality for those hazards having software causal factors.

4.2.1.8.2 Mishap Probability

The difficulty of assigning useful probabilities to faults or errors in software requires a supplemental method of determining hazard risk where software causal factors exist. Figure 4-17 demonstrates that in order to determine a mishap probability, the engineer or analyst must assess the software causal factors and the software assurance LOR processes in conjunction with the causal factors from hardware, human error, and other factors. In Figure 4-17, the probability of software's contribution to the hazard occurrence is initially unknown and is represented by question marks. Traditionally, and for the purpose of being conservative, software errors in fault trees must be set to a value of one (1) where no supporting analysis or assurance rationale is provided. This Handbook provides the process of the identification, documentation, trace,

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

implementation and tests of SSR's, and the safety assurance LOR processes and tasks for safety-significant functionality. These processes, when successfully implemented, work to reduce the probability of software's contribution to mishaps and hazards whereby the probability has to be something less than one (1) in the fault tree. Just as with hardware or human error failure mechanisms, the "perfect alignment" of circumstances can still exist for the failure initiation to occur. However, the probability of transitioning to the "perfect alignment" is what these processes work to eliminate or reduce.

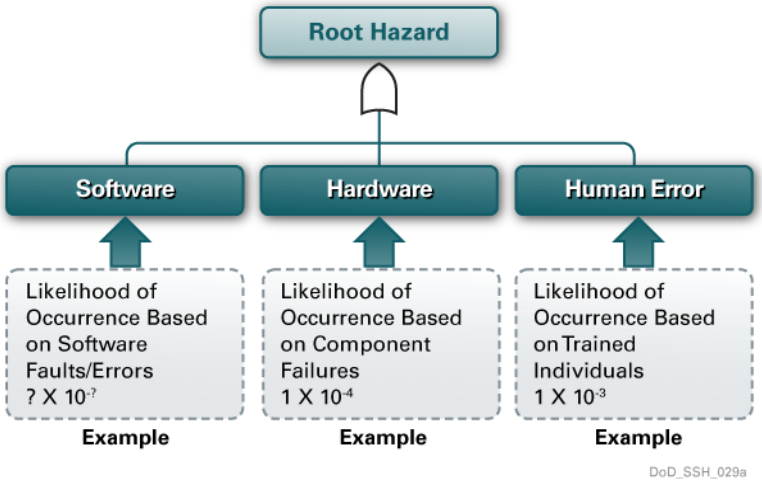

Figure 4-17: Likelihood of Occurrence Example

If the probability of software's contribution to mishap or hazard occurrence is less than the value one (1), the value used in the fault tree must be supported with sound engineering and statistical judgment, analyses, and evidences. The assignment of statistical probabilities to software failure or fault conditions that contribute to mishap or hazard occurrence is not a mature discipline. However, several programs are introducing methods to assign statistical weights to software events on the fault tree that are based upon the successful implementation of both hazard analysis processes (to specific software failure mechanisms) and the software assurance LOR tasks. This approach, while gaining acceptance in the safety community, must be supported by sound statistical methods. The software safety tasks that may specifically "drive" the statistical value down include, but are not limited to:

- Identification and traceability of safety-significant functions within the software design architecture
- The assignment of the SCI and LOR tasks associated with the requirements, design, code, and test of safety-significant functions.
- The successful implementation of the defined LOR tasks for a given safety-significant function
- Identification of specific software failure mechanisms in context with identified mishaps and hazards of the system.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- The identification and successful implementation of software requirements that mitigate the identified software causes or failure mechanisms to mishap and hazard failure conditions.
- The identification, implementation, and successful verification of software fault detection, isolation, annunciation, tolerance, and recovery requirements that are a direct result of the hazard analysis process and specific software failure contributions to mishaps and hazards.

Each identified software failure mechanism that is controlled or mitigated by design and verified and validated by inspection, analysis, and test (or other methods) helps to reduce the probability of a specific software event contributing to a mishap or hazard. In addition, each task that is successfully implemented using the LOR assurance process further reduces the likelihood of software's contribution to failure. Each task or activity accomplished can produce a "weighted" statistical value to assign probabilities to software events to the fault tree cutsets that involve software failure conditions.

The determination of hardware and human error causal factor probabilities remains constant in terms of acceptable analysis methods and historical best practices. Regardless, the risk assessment process must address the contribution of the software to the mishap or hazard's cumulative risk assessment.

4.2.2 Managing the Software Safety Program

SSS program management (Figure 4-18), like SSP management, begins as soon as the SSP is established and continues throughout the system lifecycle. Management of the effort requires a variety of tasks or processes, from establishing the SwSSWG to preparing the SAR or Safety Case. Even after a system is placed in service, management of the SSS effort continues to address modifications and enhancements to the software and the system. Changes in the use or application of a system may necessitate a re-assessment of the safety of the software in the new application.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

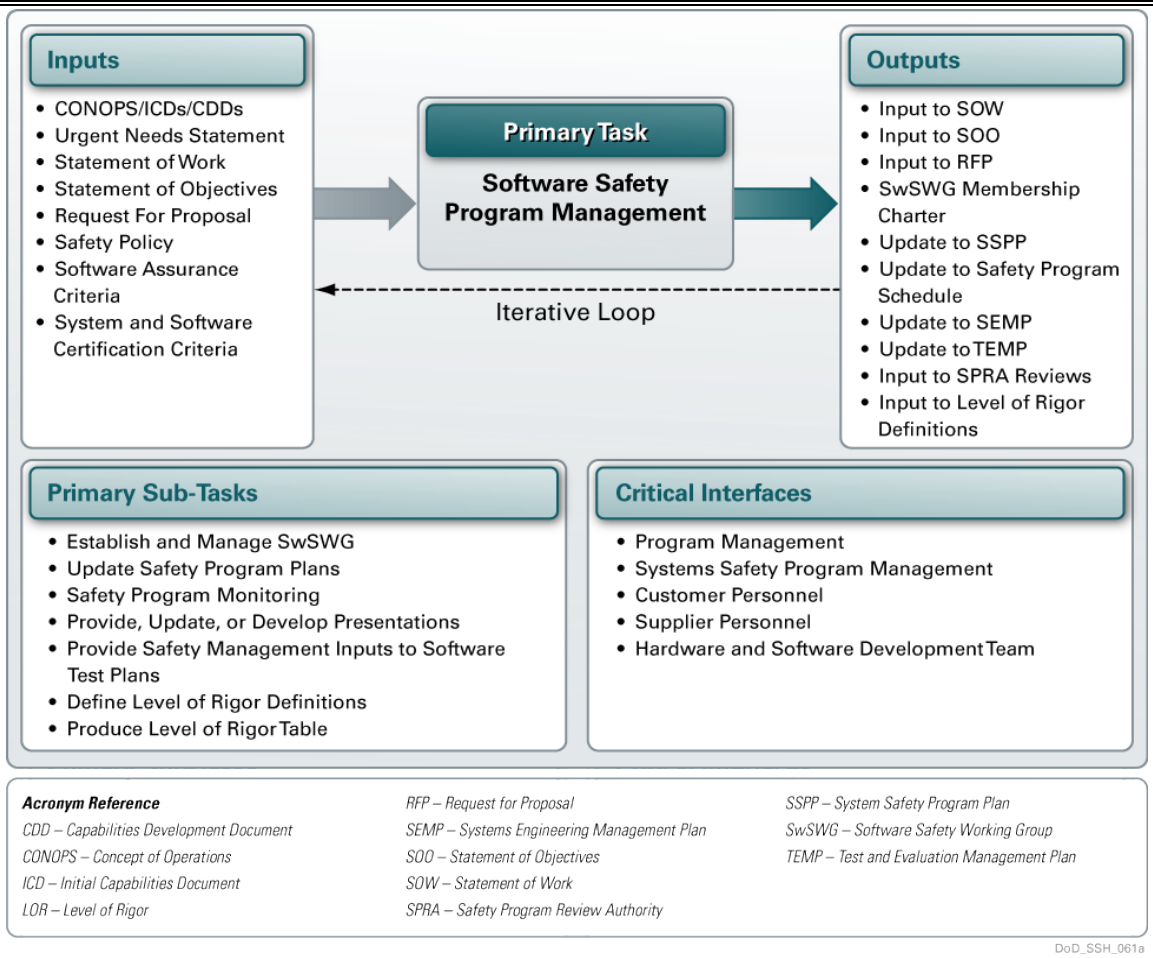

Figure 4-18: Software Safety Program Management

Effective management of the safety program is essential to the successful and efficient reduction of system risk. This section discusses the managerial aspects of the software safety tasks and provides guidance in establishing and managing an effective software safety program. Initiation of the SSP is all that is required to begin activities pertaining to software safety tasks. Initial management efforts parallel portions of the planning process since many of the required efforts (such as establishing a hazard tracking system or researching lessons learned) need to begin early in the safety program.

Safety management pertaining to software generally ends with the completion of the program and its associated testing, whether it is a single phase of the development process (e.g., concept exploration) or continues through the development, production, deployment, and maintenance phases. Management of the efforts must continue throughout the system lifecycle. From a practical standpoint, management efforts end when the last safety deliverable is completed and is accepted by the customer. Management efforts may then revert to caretaker status in which the PFS or safety manager monitors the use of the system in the field and identifies potential safety

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

deficiencies based on user reports and accident or incident reports. Even if the developer has no responsibility for the system after deployment, the safety PM can develop a valuable database of lessons learned for future systems by identifying these safety deficiencies.

Establishing a software safety program includes establishing a Software System Safety Working Group (or the software safety function within the SSWG structure). The SwSSWG is normally a sub-group of the System Safety Working Group and is chaired by the PFS or safety manager. The SwSSWG has overall responsibility for:

- Monitoring and controlling the software safety program
- Monitoring the software assurance process to ensure the completion of LOR tasks
- Identifying SSRs required by customer acceptance or certification criteria
- Identifying and resolving hazards with software causal factors
- Interfacing with the other IPTs
- Performing the final safety assessment of the system design.

A detailed discussion of the SwSSWG is located in the supplemental information of Appendix C, Section C.7.2.

It is in this early phase of the program that the Software Safety Plan of Action and Milestones (POA&M) is developed. It is integrated with the overall software development program POA&M in coordination with the system safety POA&M. Milestones from the software development POA&M, particularly design reviews and transition points (e.g., from unit code and test to integration), determine the milestones required for the software safety program. The SwSSWG must ensure that the necessary analyses are complete in time to provide input to development efforts to ensure effective integration of software safety into the overall software development process.

One of the most difficult aspects of software safety program management is the identification and allocation of resources required to adequately assess the safety of the software. In the early planning phases, the configuration of the system and the degree of interaction of the software with the potential hazards in the system are largely unknown. The higher the degree of software involvement, the greater the resources required to perform the assessment. To a large extent, the software safety Program Manager can use the early analyses of the design, participation in the functional allocation, and high-level software design to ensure that the amount of safety-significant software is minimized. If safety-critical functions are distributed throughout the system and related software, the software safety program must encompass a much larger portion of the software. However, if the safety-critical functions are associated with as few software modules as practical, the level of effort may be significantly reduced.

Effective planning and integration of software safety efforts into other IPTs will significantly reduce the software safety-significant tasks that must be performed by the SSS team. Incorporating the generic SSRs into the plans and specifications developed by other IPTs allows

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

them to assume responsibility for their assessment, performance, and evaluation. For example, if the SSS team provides the quality assurance generic SSRs to the SQA IPT, the SSS team will perform compliance assessments with requirements, not just for safety, but for all aspects of the software engineering process. In addition, if the SQA IPT "buys into" the software safety program and its processes, it supplements the efforts of the software safety engineering team, reduces workload, and avoids duplication of effort. The same is true of other IPTs, such as Configuration Management and Software Test and Evaluation. In identifying and allocating resources to the software safety program, the software safety Program Manager can perform advance planning, establish necessary interfaces with other IPTs, and identify individuals to act as software safety representatives on those IPTs.

Identifying the number of analyses and the level of detail required to adequately assess the software involves a number of processes. Experience with prior programs of a similar nature is the most valuable resource that the software safety PM has for this task. However, every program development is different and involves different teams of people, PA requirements, and design implementations.

The process begins with the identification of the preliminary system-level hazards in the PHL. This provides initial safety concerns that must be assessed in the overall safety program. From the system specification review process and the FHA, the functional allocation of requirements results in a high-level distribution of safety-critical functions and system-level safety requirements to the design architecture. Software functions that have a high safety-criticality (e.g., warhead arming and firing) will require a significant analysis effort that may include code-level analysis. Safety involvement early in the design process can reduce the amount of software that requires analysis; however, the software safety manager must still identify and allocate resources to perform these tasks. Safety requirements that conflict with others (e.g., reliability) require trade-off studies to achieve a balance between desirable attributes.

The software control categories discussed in Section 4.2.1.4, Table 4-1, provide a useful tool for identifying software that requires high levels of analysis and testing. The more critical the software, the higher the level of effort necessary to analyze, test, and assess the risk associated with the software. In the planning activities, the SwSSWG identifies the analyses necessary to assess the safety of specific modules of code. Experience is the best teacher for determining the level of effort required. These essential analyses do not need to be performed by the software engineering group and may be assigned to another group or person with the necessary specialized expertise. The SwSSWG will have to provide the needed safety-significant guidance and training to the individuals performing the analysis.

The most important aspects of software safety program management are monitoring the activities of the safety program throughout system development to ensure that tasks are on schedule and within cost and identifying potential problem areas that could impact the safety or software development activities. The software safety manager must:

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Monitor the status and progress of the software and system development effort to ensure that program schedule changes are reflected in the software safety program POA&M.
- Monitor the progress of the IPTs and ensure that the safety interface for each is working effectively. When problems are detected, either through feedback from the software safety representative or other sources, the software safety manager must take the necessary action to document and mitigate the problem.
- Monitor and receive updates regarding the status of analyses, open hazard records, and other safety activities on a weekly basis. Significant hazard records should be discussed at each SwSSWG meeting and updated as required. A key factor that the software safety Program Manager must keep in mind is the tendency for many software development efforts to begin compressing the test schedule as slippage occurs in the software development schedule. The software safety Program Manager must ensure that the safety test program is not compromised as a result.

The contract should identify the safety review requirements; however, it is the responsibility of the DA to ensure that the software safety program incorporates the appropriate reviews into the SwSPP. The system safety manager must identify the appropriate SRB and review the schedule during the development process. SRBs generally involve significant effort outside of other software safety tasks. The DA must determine the level of effort and support required for each review and incorporate this information into the SwSPP. For complex systems, multiple reviews may be required to update the SRB documentation and ensure that all PA requirements are achieved.

Although SRB requirements may vary from each PA, some require a technical data package (TDP) and briefing to a review board. The technical data package may be a SAR or may be considerably more complex. The DA must determine whether the TDP and briefing need to be provided, or whether that activity is to be performed independently. In either event, safety program personnel may be required to participate in or attend the reviews to answer specific technical questions. The presenters usually require several weeks of preparation for the SRBs. Preparation of the TDP and supporting documentation requires time and resources even if the data package is a draft or final version of the SAR.

4.3 Software Safety Task Implementation

This section of the Handbook describes the primary task implementation steps required for a baseline SSS engineering program. This section presents the tasks required for the integration of software safety activities into the functional areas of system and software development. The software system safety process includes two specific complementary processes—the hazard analysis process that specifically identifies software causal factors and the software assurance and integrity process that includes the level of rigor required for the design, implementation, and test of safety-significant functions. These two processes focus heavily on the identification, implementation, and verification of the SSRs to reduce the likelihood of a software contribution

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

to hazards and mishaps. While the specific tasks are presented in series, the actual implementation of many of these steps is accomplished concurrently.

As the Handbook introduces the software safety engineering process, it will identify inputs to the described tasks and the products that each specific process step produces. Each program and engineering interface tied to software safety engineering must agree with the processes, tasks, and products of the software safety program and with the timing and scope of effort to verify that it is in concert with the objectives and requirements of each interface. If individuals in other program disciplines are not in agreement or do not see the functional utility of the effort, these individuals will usually default to a "non-support" mode.

Figure 4-19 provides a depiction of the software safety activities required for the implementation of a credible SSS program. Remember that the process steps identified in this Handbook represent a baseline program that has a base in historical lessons learned and include the best practices from successful programs. Because each procurement, software acquisition, or development has the potential and probability to be uniquely diverse, the safety manager must use this paragraph only as a guide. Each of the following steps should be analyzed and assessed to identify where minor changes are required or warranted for the software development program proposed. If these tasks, with the implementation of minor changes, are incorporated in the system acquisition lifecycle, the SSS effort has a very high likelihood of success.

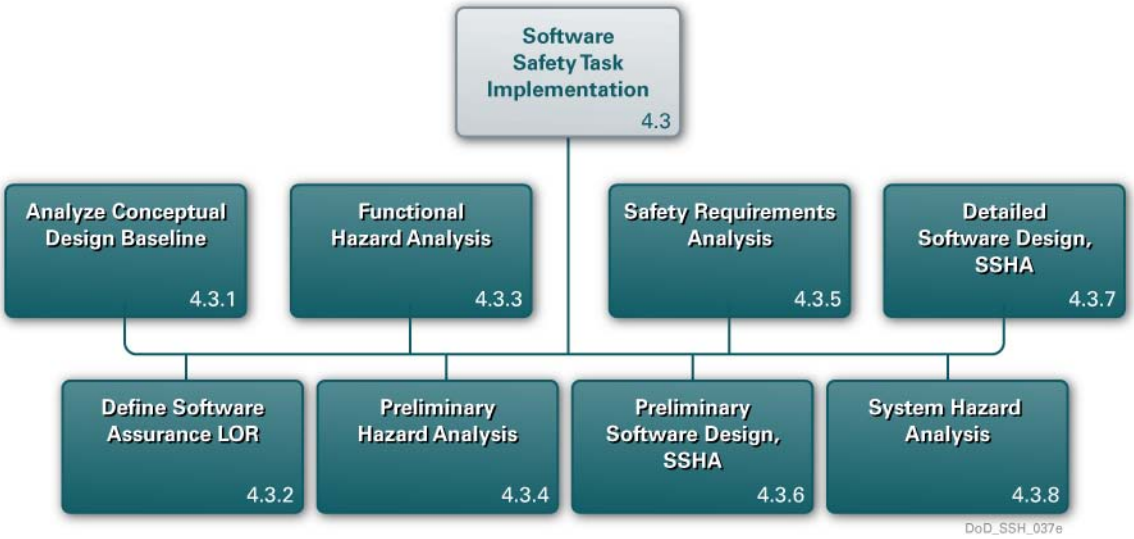

Figure 4-19: Software Safety Task Implementation

The software safety engineering activities within the hardware and software development project depends on the individual(s) performing the managerial and technical safety tasks. The success of the program depends on the identification of a logical, practical, and cost-effective process that produces the products to meet the safety objectives of the program. The primary safety products include hazard analyses, initial safety design requirements, functionally-derived safety

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

design requirements (based on hazard causes), test requirements to produce evidence for the elimination or control of safety hazards, and the identification of safety requirements pertaining to operations and support of the product. The managerial and technical program leads must agree that the software safety tasks defined in this paragraph will provide the documented evidence for the resolution of identified hazards and failure modes in design, implementation (code), fabrication, test, deployment, and lifecycle support activities. Residual safety risk must be thoroughly defined and communicated to program management during each phase of the lifecycle.

Planning and management of a successful software safety program is aligned and integrated with the safety engineering and management program schedule. Safety schedules should include near-term and long-term events, milestones, and contractual deliverables. The schedule should also reflect the system safety management and engineering tasks that are required for each lifecycle phase of the program and that are required to support program milestone decisions. Specific safety data to support special safety boards or safety studies for compliance and certification purposes is also crucial. Examples include FAA certification, U.S. Navy Weapon Systems Explosives Safety Review Board approval, Defense Nuclear Agency Nuclear certification, and U.S. Air Force Non-Nuclear Munitions Safety Board approval. The PM must track each event, deliverable, and milestone to ensure that safety analysis activities are timely to facilitate cost-effective and technically feasible design solutions. These activities ensure that the SSS program will meet the desired safety specifications of program and system development activities.

Figure 4-20 provides an example milestone schedule for a software safety program. This figure depicts the relationship of safety-specific activities to the acquisition lifecycles of both system and software development. While the figure appears as a Waterfall lifecycle model, the figure depicts when specific system safety tasks are likely to be accomplished against major milestone schedules. Each program must integrate the individual tasks and deliverables of their program to the acquisition lifecycle model used for that program.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

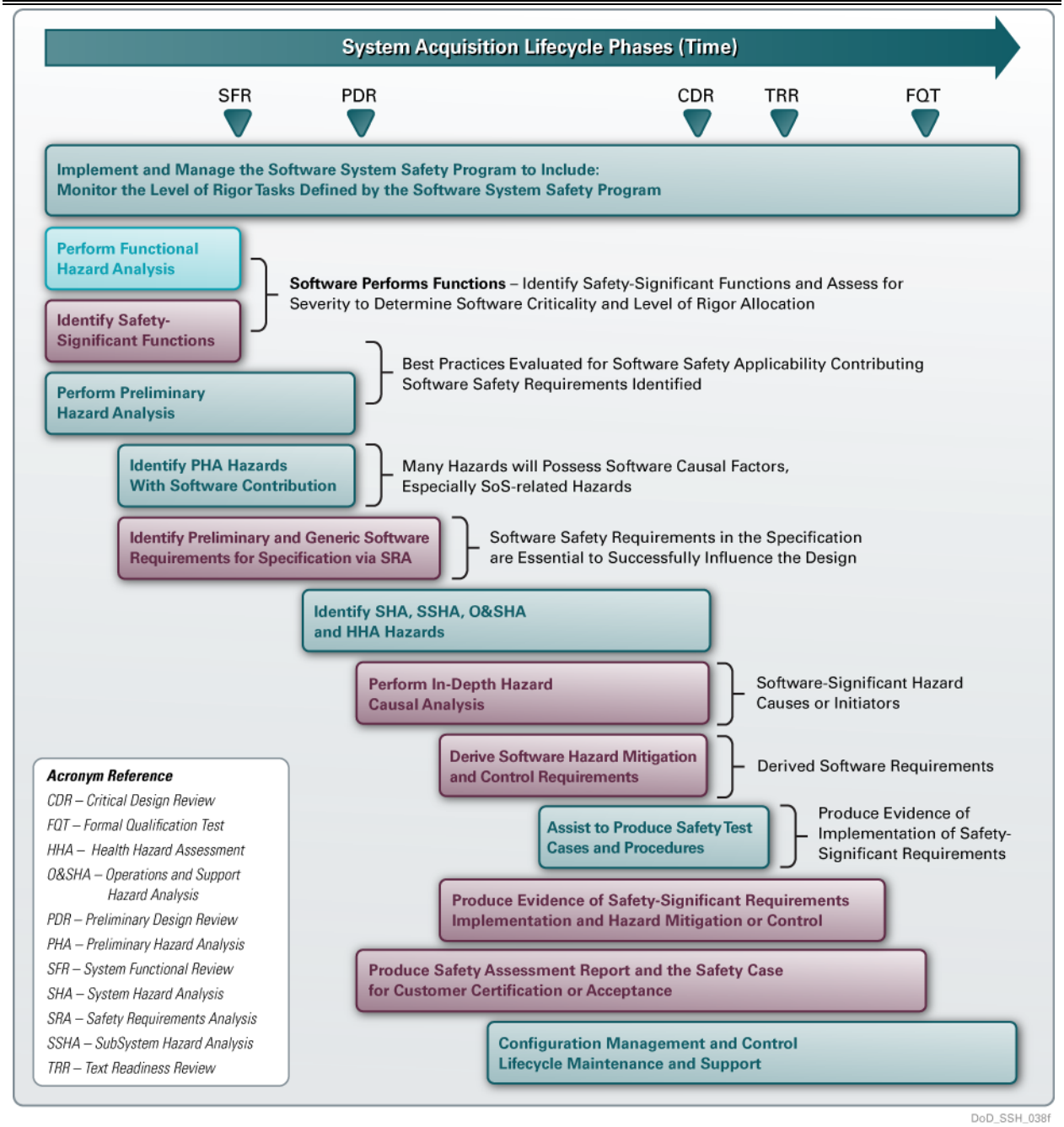

Figure 4-20: Example POA&M Schedule

Remember that each procurement is unique and will have subtle differences associated with managerial and technical interfaces, timelines, processes, and milestones. This schedule is an example with specific activities and time relationship-based typical programs. Program planning must integrate program-specific differences into the schedule and support the practical assumptions and limitations of the program. While the specific tasks of the system safety

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

program are presented in series, the actual implementation of these tasks is accomplished concurrently.

As described in Section 4.2.2, the POA&M will also include the safety reviews, PA reviews, internal reviews, and SwSSWG meetings. The software safety assessment milestones are generally geared to the SRBs since the technical data package required is either the draft or final software-significant SAR. Hazard analysis schedules must reflect program milestones where hazard analysis input is required. For example, SSRs resulting from generic requirements tailoring (documented in the SRA) must be available as early as practical in the design process for integration into design, programmatic, and system safety documents. Specific safety requirements from the PHA and an initial set of safety design requirements must be available prior to the Preliminary Design Review (PDR) for integration into the design documents. System safety and software safety must participate in the system specification review and provide recommendations during the functional allocation of system requirements to hardware, software, operation, and maintenance.

After functional allocation is complete, the Software Engineering IPT, with the help of the software safety member, will develop the SRS. At this point, SSS should have the preliminary software safety assessment complete, with hazards and safety-significant functions identified and safety-significant functions allocated to the software design team. The SwSSWG updates the analyses as the system development progresses; however, the safety design requirements (hardware, software, and human interfaces) must be complete prior to the Critical Design Review (CDR). Requirements added after the CDR can have a major impact on program schedule and cost.

The development of safety test requirements begins with the identification of SSRs. SSRs can be safety contributing requirements, generic requirements, or functional (derived) mitigating requirements generated from the implementation of hazard controls (discussed in more detail in Section 4.3.5). SSRs incorporated into software documentation automatically become part of the software test program. Throughout development, the software safety organization must ensure that the test plans and procedures will provide the desired validation of SSRs in accordance with their assigned LORs, demonstrating that they meet the intent of the requirement. Section 4.4 provides additional guidance on the development of the safety test program. Detailed inputs regarding specific safety tests are derived from the hazard analyses, causal factor analysis, and the definition of software hazard mitigation requirements. Safety-specific test requirements are provided to the test organization for development of specific test procedures to validate the SSRs. The analysis associated with this phase begins as soon as test data from the safety tests is available.

The SHA begins as soon as functional allocation of requirements occurs and continues through the completion of system design. Specific milestones for the SHA include providing safety test requirements for integration testing to the test organization and detailed test requirements for interface testing. The latter will be required before testing of the software with other system components begins.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The SAR is written throughout the lifecycle of the program to provide a snapshot of the level of safety that has been achieved at each major milestone. These milestones include major test events, program milestones, and prior to presenting the program to the appropriate certification review boards.

4.3.1 Analyze and Comprehend the Conceptual Design Baseline of the System

Optimally, the software safety tasks begin as soon in the acquisition lifecycle development process as possible. In reality, there will be times that participation on an individual program will be further into the schedule than desired. Regardless of when participation begins, the initial task involves gathering required data, information, and documentation for the initial assessment of the system. This assessment requires the system safety engineer to analyze and comprehend numerous aspects of the system and its intended operational environments. Specific information to be reviewed and assessed includes, but is not limited to:

- Initial Capabilities Document (ICD) or Capability Development Document (CDD)
- Concept of operations
- Product specification
- System specification
- Functional specification
- Software specification
- Trade studies
- Design engineering drawings (or presentation material)
- Operational view
- Interface Specification, ISP, or IER.

The safety engineer must be diligent in this task to ensure that complete and accurate descriptions of the physical and functional aspects of the system are collected for initial safety analysis. In addition to the description of the system, it is important to comprehend the user requirements for the system and the operational environments in which the system will be deployed. These environments may be contributing factors in mishap and hazard scenarios that will be considered in the safety analyses.

4.3.2 Define Software Assurance Levels of Rigor

The second step in the implementation of a software safety engineering program is to define the software assurance LOR tasks to be accomplished in the design, implementation, and test of safety-significant functions. The specific content elements for the LOR table are described in Section 4.2.1.5.This task begins with the identification of specific terms and definitions to be used for the program as it relates to a credible software safety assurance program. The customer and the supplier must specifically define and agree on the terms to be used and the definitions of these terms. The items and definitions must be formally documented in the SSMP/SSPP and

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

included in the SDP and STP. If program management desires to change the verbiage of the software control categories, they must also be defined and documented at this time. Specific definitions of safety-related, safety-critical, and safety-significant terms must be included in the SSPP. Software control category definitions were previously provided in Table 4-1.

The customer and supplier must define and agree upon the specific LOR categories and the number of levels to be used in the program. Optimally, this task should be accomplished as part of the RFP/SOW and the proposal/contract portion of the program. This task should be finalized in agreement as part of the SSPP submittal and approval process. As previously depicted in Figure 4-16, the example LOR table includes five specific safety risk categories ranging from High risk to very Low risk, where the example Table 4-4 depicted only three categories of safety risk.

4.3.3 Functional Hazard Analysis

The benefits of performing an FHA (Figure 4-21) as early as possible in the acquisition lifecycle are numerous. The objectives and corresponding benefits include, but are not limited to:

- Provide an early and complete understanding of the physical attributes of the system and its intended functionality, logical structure, and data attributes
- Identify each system function for categorization and prioritization for functions that are safety significant (both safety critical and safety related)
- Map safety-significant functions to the physical design and develop a means to identify all safety-critical or safety-related components
- Map safety-significant functions to the software design and develop a means to identify safety-critical or safety-related modules of code and where the functions reside in the software architecture
- Map safety-significant functions to the human interfaces of the system
- Map safety-significant functions to other interfaces and systems in the SoS
- Assist in the understanding of the interfaces between hardware, software, and the control entity
- Provide a mechanism to identify top-level mishaps (TLMs), system hazards, and subsystem hazards which supplement the PHL and the PHA. NOTE: While DoD uses the PHL and PHA as the mechanism to define mishaps and hazards, agencies and contractors using SAE ARP 4754 and 4761 may be using the FHA as their only mechanism to identify hazards
- Identify the rationale for preliminary (generic) top-level safety requirements for the hardware and software specifications
- Provide confidence to the customer that all hardware, software, and human functionality is accounted for in the safety analysis
- Provide the customer with safety drivers at interfaces outside the system.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

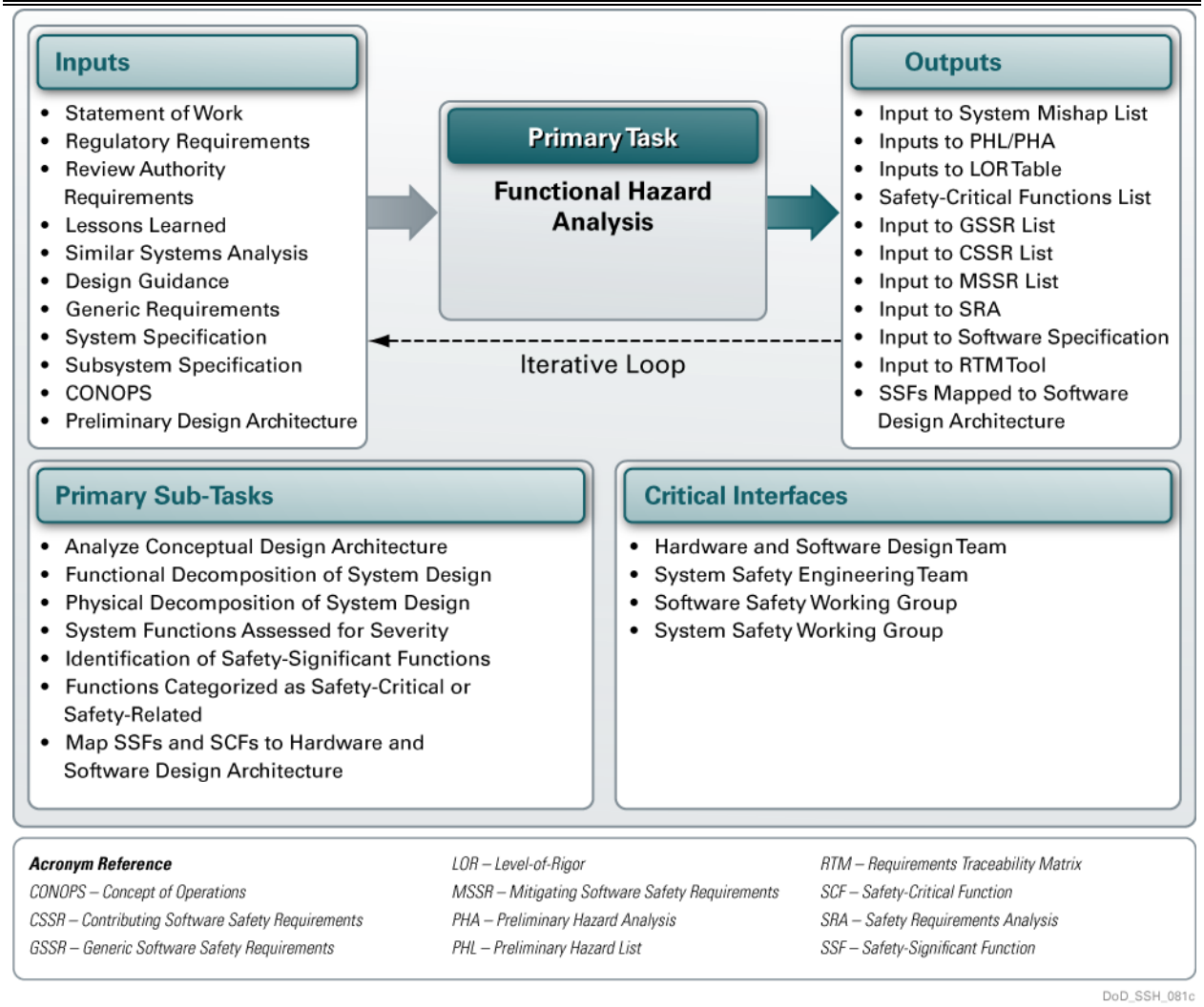

Figure 4-21: Functional Hazard Analysis

When performed early in the acquisition lifecycle, the effort begins with the safety engineer analyzing the functionality of each segment of the conceptual design. From the list of system functions in the functional specification, the engineer must determine the safety severity ramifications of loss of function, interrupted function, incomplete function, function occurring out of time or sequence, or function occurring inadvertently. There are times, however, when the safety engineer is brought onto the project late. If the FHA is performed later in the acquisition lifecycle, the safety engineer may begin the analysis by accounting for each major subsystem and its hardware components and then identifying what each does functionally.

Regardless of the approach, the FHA activity provides for the initial identification of safety-significant functions. The rationale for the identification of safety-significant functions of the system is addressed in the identification of safety deliverables (Appendix C, Section C.1.4). This activity must be performed as a part of the defined software safety process to ensure that the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Project Manager, systems and design engineers, software developers, and test engineers are aware of each safety-significant or critical function of the design. This process also ensures that each individual module of code that performs these functions is officially labeled as "safety significant" (either safety critical or safety related) and that defined levels of design and code analysis and test activity are mandated in the approved LOR table. An example of the possible safety-critical functions of a tactical aircraft is provided in Figure 4-22.

SAFETY-CRITICAL FUNCTIONS
**** for example purposes only ****

- Altitude Indication
- Attitude Indication
- Air Speed Indication
- Engine Control
- In-Flight Restart After Flameout
- Engine Monitor and Display
- Bleed Air Leak Detection
- Engine/APU Fire Detection
- Fuel Feed for Main Engines
- Fire Protection/Explosion Suppression
- Flight Control—Level III Flying Qualities
- Flight Control—Air Data
- Flight Control—Pitot Heat
- Flight Control—Fuel System/CG Control
- Flight Control—Cooling
- Flight Control—Electrical
- Flight Control—Hydraulic Power
- Canopy Defog
- Adequate Oxygen Supply to the Pilot
- Stores and Expendables Separation
- Safe Gun and Missile Operation
- Armament/Expendables for System Ground Operations
- Emergency Canopy Removal
- Emergency Egress
- Ejection Capability
- Landing Gear Extension
- Ground Deceleration
- Structure Capability to Withstand Flight Loads
- Freedom From Flutter
- Stability in Pitch, Roll, and Yaw
- Heading Indication
- Fuel Quantity Indication
- Fuel Shut-off to Engine and APU
- Engine Anti-Ice
- Caution and Warning Indications

DoD_SSH_039c

Figure 4-22: An Example of Safety-Critical Functions

SAE ARP 4761 provides a reasonable description of how to perform and what to include in a typical FHA, but remember that this specific standard is aircraft-centric and each system may require a different viewpoint (e.g., submarine, mining, or automobile). Table 4-5 represents an FHA template table that can be used to accomplish the analysis. Using this table as an example, there may be a "one-to-many" relationship between components and functions. That is, components in the system can have more than one functional purpose. Each function must be accounted for and analyzed. Each function allocated to the software design will be assessed in terms of SCC and assigned a LOR based on its safety criticality within the system.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table 4-5: Example FHA Template

WBS NO.	Subsystem or Major Component	Functionality	Interfaces	Consequence of Loss of Function	Severity of Consequence	Safety-Significant Functions	Mishaps or Hazards	Safety-Significant Software Modules
1.0	Major Subsystem							
1.1	Component							
1.2	Component							
1.n	Component							
2.0	Major Subsystem							
2.n	Component							
n.0	Major Subsystem							

DoD_SSH_077a

Each analyst may set up the FHA template differently based on the information to be included in the worksheet for analysis. Regardless of how the template is set up, the defined objectives of the FHA must be fulfilled. This task "kick-starts" both the LOR software assurance tasks and the software hazard analysis activities.

From a software safety perspective, the FHA is important to identify safety-significant functions, assess these functions for the safety severity of the consequence of losing or having the function degrade, and map these functions to the software design architecture. From a system safety perspective, a byproduct of the FHA is the insight required to identify mishaps and hazards for the PHL and PHA. The probability of a safety-significant function failing is not included in the FHA unless specifically defined by contract.

The FHA can include other information predicated on either SAE ARP 4761 or specific information that is considered useful and available for analysis. Information (additional columns) can be added from a SoS perspective as to whether information produced by a component (or function) is provided to other users or obtains input for use by outside users.

4.3.4 Preliminary Hazard Analysis

4.3.4.1 PHL Development

The PHL is described in Appendix C, Section C.3.3. In most instances the PHL will precede the PHA and is usually contract and technology-maturity dependent. The PHL is the initial set of

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

potential hazards associated with the system under development. Development of the PHL requires knowledge of the physical and functional requirements of the system and some foreknowledge of the conceptual system design. The documentation of the PHL helps initiate the analyses that must be performed on the system, subsystems, and interfaces. The PHL is based on the review of analyses of similar systems, lessons learned, potential kinetic energies associated with the design, design handbooks, and user and systems specifications. The generated list also aids in the development of initial (or preliminary) requirements for the system designers and the identification of programmatic (technical or managerial) risks to the program.

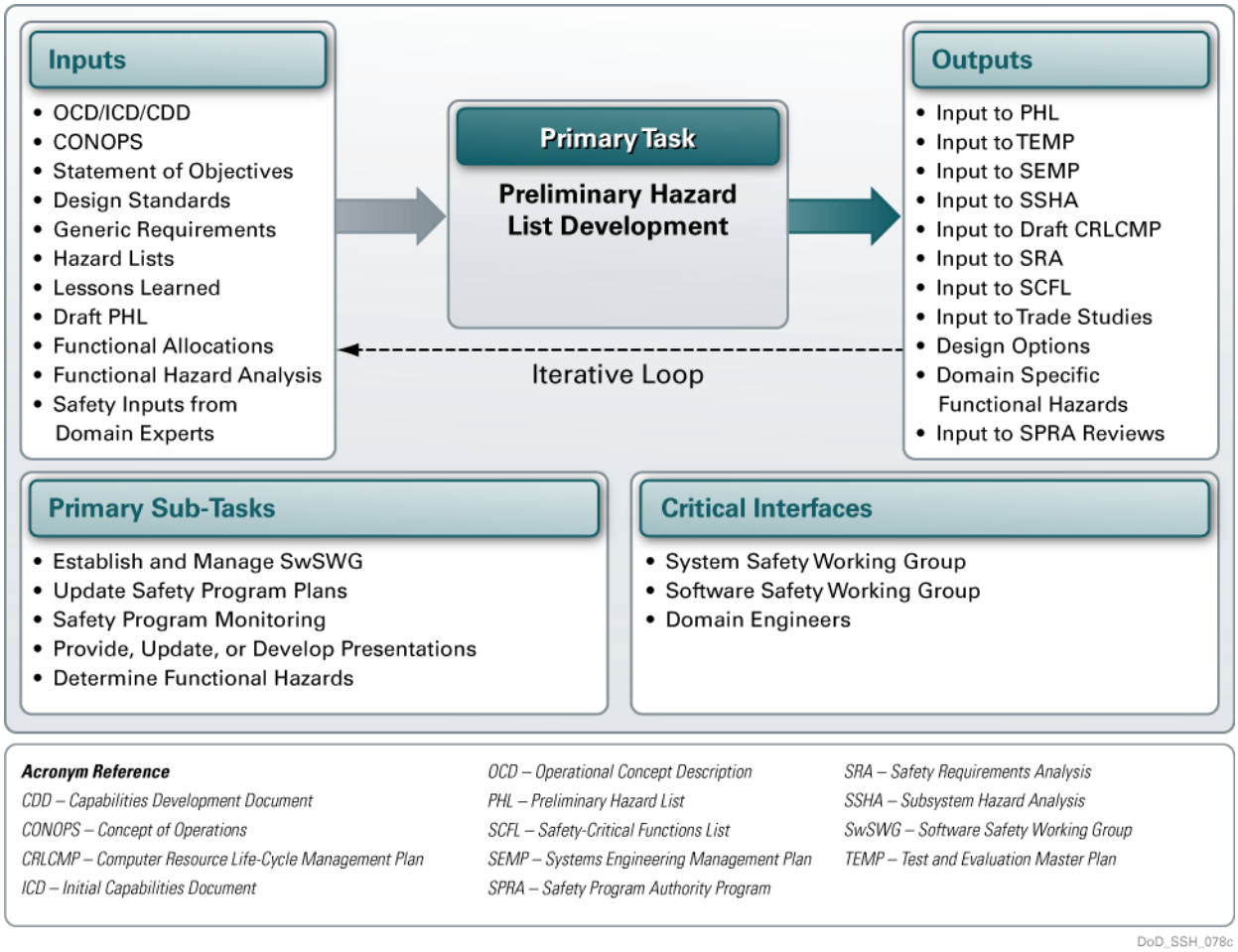

Figure 4-23: PHL Development

The PHL (Figure 4-23) is an integrated engineering task that requires cooperation and communication between functional disciplines and among operational users; maintainers; and the systems, safety, and design engineers. The assessment and analysis of all preliminary and current data pertaining to the proposed system accomplish this task. From a documentation perspective, the following should be available for review:

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- ICD or CDD
- Preliminary system specification
- Preliminary product specification
- User requirements document
- Lessons learned
- Analysis of similar systems
- Prior safety analyses (if available)
- Design criteria, handbooks, and standards
- Functional Hazard Analysis.

From the preceding list of documentation and functional specifications, the system safety engineer develops a preliminary list of TLMs and system hazards for further analysis. Although the identified hazards may appear to be general or preliminary at this time, this is normal for the early phase of system development. As the hazards are analyzed against system physical and functional requirements, they will mature to become the hazards fully documented in the PHA, SSHA, SHA, and O&SHA and correlated to the TLMs of the system. A preliminary risk assessment of the PHL hazards will help determine whether trade studies or design options must be considered to reduce the potential for unacceptable or unnecessary safety risk in the design.

In addition to the information assessed from preliminary documents and databases, technical discussions with systems engineering help determine the ultimate safety-significant functions associated with the system. Functions that should be assessed include manufacturing, fabrication, operations, maintenance, and test. Other technical considerations include transportation and handling, software/hardware interfaces, software/human interfaces, hardware/human interfaces, environmental health and safety, explosive and other energetic components, product loss prevention, and nuclear safety.

At this phase of the program, specific ties from the PHL to the software design are premature and are generally based on knowledge and lessons learned from similar systems. Specific ties to the software are normally through hazard causal factors, which have yet to be defined at this point in the development. However, there may be identified hazards with preliminary ties to safety-significant functions which are functionally linked to the preliminary software design architecture. If this is the case, this functional link should be adequately documented in the safety analysis for further development and analysis. At the same time, there are likely to be generic SSRs applicable to the system (see Appendix E). These requirements are available from multiple sources and must be specifically tailored to the program as they apply to the system design architecture.

4.3.4.2 PHA Development

The PHA is a safety engineering and software safety engineering analysis performed to identify and prioritize the preliminary TLMs, hazards, and their causal factors in the system under development. Figure 4-24 depicts the safety engineering process for the PHA. Many safety

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

engineering texts provide guidance for developing the PHA. This Handbook will not detail these processes. Each methodology focuses on a process that will identify a substantial portion of the hazards; however, none of the methodologies are complete. Many analysts use the lifecycle profile of a system as the basis for the hazard identification and analysis. Unless the analyst is particularly astute, subtle system hazards and causal factors may be missed.

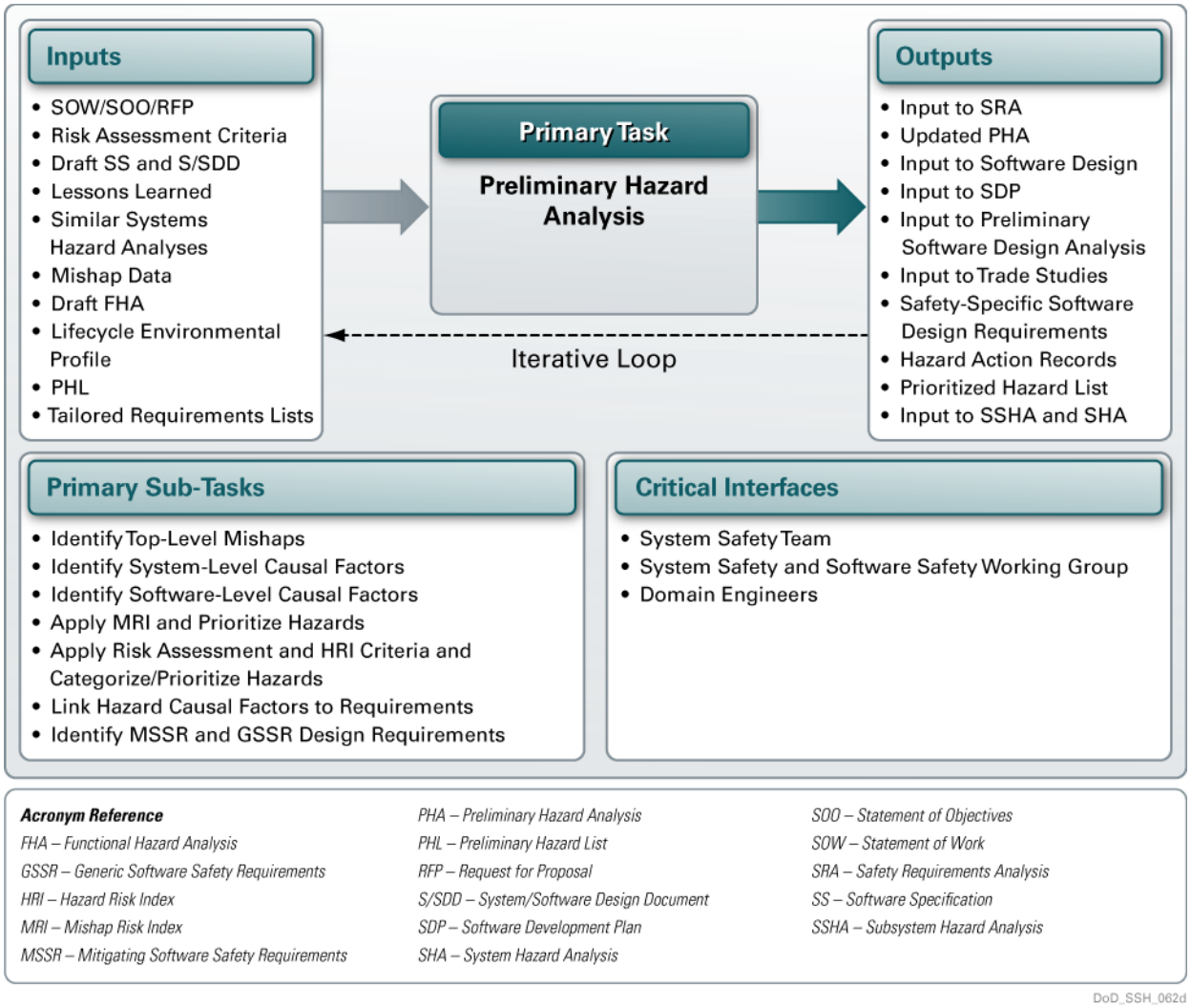

Figure 4-24: PHA

The PHA is the springboard analysis to launch the SSHA and SHA analyses as the design matures and progresses through the development lifecycle. Preliminary hazards can be eliminated (or officially closed through the SSWG) if they are deemed to be inappropriate for the design. Remember that this analysis is preliminary and is used to provide early design considerations that may or may not be derived or matured into design requirements.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Throughout this analysis, the PHA provides input to trade-off studies. Trade-off analyses performed in the acquisition process are listed in Table 4-6. These analyses offer alternative considerations for performance, producibility, testability, survivability, compatibility, supportability, reliability, and system safety during each phase of the development lifecycle. System safety inputs to trade studies include the identification of potential or real safety concerns and the recommendations of credible alternatives that may meet all (or most) of the requirements while reducing overall safety risk.

Table 4-6: Acquisition Process Trade-Off Analyses

Acquisition Phase	Trade-Off Analysis Function
Mission Area Analysis	Prioritize Identified User Needs
Material Solution	Compare New Technologies with Proven Concepts • Select Concepts Best Meeting Mission Needs • Select Alternative System Configuration
Technology Development	Select Technology • Reduce Alternative Configurations to a Testable Number
Engineering and Manufacturing Development	Select Component/Part Designs • Select Test Methods • Select Operational Test and Evaluation Quantities
Production and Deployment	Compare New Technologies with Proven Concepts • Perform Make-or-Buy, Process, Rate, and Location Decisions

DoD_SSH_079

The entire unabridged list of potential hazards developed in the PHL is the entry point of the PHA. The list should be checked for applicability and reasonability as the system design progresses. The first step is to eliminate any hazards from the PHL that are not applicable to the system (e.g., if the system uses a Linux operation system, eliminate any hazards specific to other operating systems (OSs)). The next step is to categorize and prioritize the remaining hazards according to the (system) RAC. The categorization provides an initial assessment of system mishap severity and probability of occurrence, and thus, the safety risk. The probability assessment at this point in the process is usually subjective and qualitative. After completion of a prioritized list of preliminary hazards, the analysis continues with the determination of the hardware, software, and human interface causal factors in context to the individual hazards as shown in the example in Figure 4-25.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

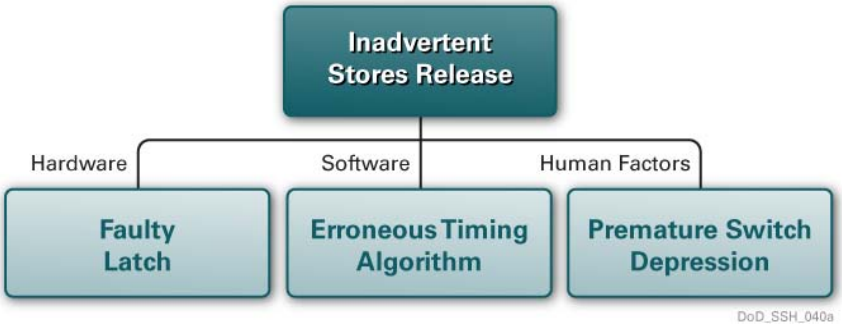

Figure 4-25: Hazard Analysis Segment

This differentiation of causes assists in the separation and derivation of specific design requirements for implementation in software. For example, as the analysis progresses, the analyst may determine that software or hardware could subsequently contribute to a hardware casual factor. A hardware component failure may cause the software to react in an undesired manner, leading to a hardware-influenced software causal factor. The analyst must consider all paths and all hardware, software, and human contribution to failure to ensure coverage of the software safety analysis.

Although this tree diagram can represent the entire system, software safety is particularly concerned with the software causal factors linked to individual hazards and ensuring that the mitigation of each causal factor is traced from requirements to design, code, and test. These preliminary analyses and subsequent system and software safety analyses identify when software is a potential cause or contributor to a hazard or will be used to support the control of a hazard.

At this point, tradeoffs evolve. It should become apparent at this time whether hardware, software, or human procedures and training best mitigate the first-level causal factors of the PHL item (the root event that is undesirable). This causal factor analysis provides insight into the best functional allocation within the software design architecture. Requirements designed to mitigate the hazard causal factors do not have to be one-to-one (e.g., one software causal factor does not necessarily yield one software control requirement). Safety requirements can be one-to-one, one-to-many, or many-to-one in terms of controlling hazard causal factors to acceptable levels of safety risk.

In many instances, designers can use software to compensate for hardware design deficiencies or where hardware alternatives are impractical. Because software is perceived to be cheaper to change than hardware, software design requirements may be specified to control specific hardware causal factors. In other instances, one design requirement (hardware or software) may eliminate or control numerous hazard causal factors (e.g., some generic requirements). This illustrates the importance of not accomplishing hardware safety analysis and software safety analysis separately.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

A system-level or subsystem-level hazard can be caused by a single causal factor or a combination of many causal factors. The safety analyst must consider all aspects of what causes the hazard and what will be required to eliminate or control the hazard. Hardware, software, and human factors cannot usually be segregated from the hazard and cannot be analyzed separately. The analysis performed at this level is integrated into the trade-off studies to allow programmatic and technical risks associated with various system architectures to be determined.

Both software-initiated causes and human error causes influenced by software input must be adequately communicated to the systems engineers and software engineers to identify software design requirements that preclude the initiation of the root hazard identified in the analysis. The software development team may have already been introduced to the applicable generic SSRs. These requirements must address how the system will safely react to operator errors, component failures, functional software faults, hardware/software interface failures, and data transfer errors. As detailed design progresses, functionally-derived software requirements will be defined and matured to specifically address causal factors and failure pathways to a hazardous condition or event. Communication with the software design team is paramount to ensuring adequate coverage in preliminary design, detailed design, and testing.

If a PHL is executed on a system that has progressed past the requirements phase, a list or a tree of identified software safety-critical functions becomes helpful to flesh out the fault tree or the tool used to represent the hazards and their causal factors. The fault tree method is one of the most useful tools in the identification of specific causal factors in both hardware and software.

During the PHA activities, the link from the software causal factors to the system-level requirements must be established. If there are causal factors that cannot be linked to a requirement when descriptively inverted, they must be reported back to the SSWG for additional consideration. This may require development and incorporation of additional requirements or implementations into the system-level specifications.

The hazards are formally documented in a hazard tracking database record system. The entries include information regarding the description of the hazard, causal factors, the effects of the hazard (possible mishaps), and the preliminary design requirements for hazard control. Controlling causal factors reduces the probability of occurrence of the hazard. Performing the analysis includes assessing hazardous components, safety-significant interfaces between subsystems, environmental constraints, operation, test and support activities and facilities, emergency procedures, and safety-significant equipment and safeguards. A suggested PHA format (Figure 4-26) can be specified via the SOW or the CDRL, and its corresponding information can be included in the hazard tracking database. This is only a summary of the analytical evidence that needs to be progressively included in the SDL to support the final safety and residual risk assessment in the SAR.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

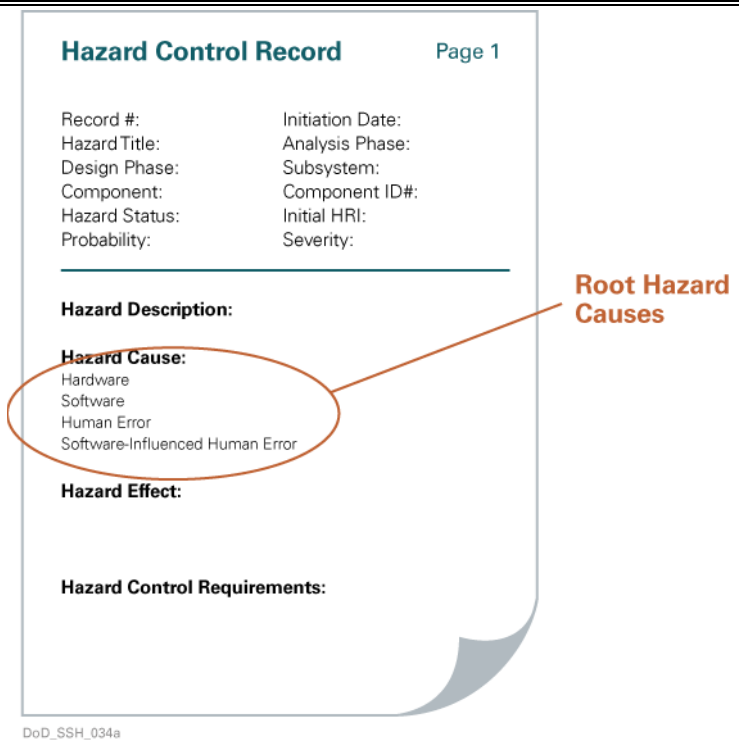

Figure 4-26: Example of a PHA Format

The PHA becomes the input document and information source for all other hazard analyses performed on the system, including the FHA, SSHA, SHA, and the O&SHA.

4.3.5 Safety Requirements Analysis

The identification of safety requirements for the system or software specification is one of the most important tasks that a safety engineer will perform. All hazard analysis activities include the identification of safety requirements of the system to mitigate safety risk potential. The safety requirements are the driving force behind a designer's ability to design safety into a system and its subsystems. An SRA is performed to document the safety design requirements and the criteria that justify the safety requirements for a system under development.

From a safety perspective, there are three categories of SSRs that will be analyzed in the SRA and documented and "tagged" as safety-significant in the SRS. They are contributing software safety requirements (CSSR), generic software safety requirements, and mitigating software safety requirements (MSSR). Software safety requirements can be any of the three categories and are derived from known safety-critical functions, tailored best practices, and hazard causal factors determined from previous activities. Figure 4-27 identifies the software safety engineering process for developing the SRA.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

From a programmatic perspective, these three categories allow discussion with the systems engineers, software engineers, and test engineers regarding how to handle the implementation and measurement of the requirement. For example, contributing requirements will need to be designed to minimize the contribution, while mitigation requirements need to be tested to validate minimized risk.

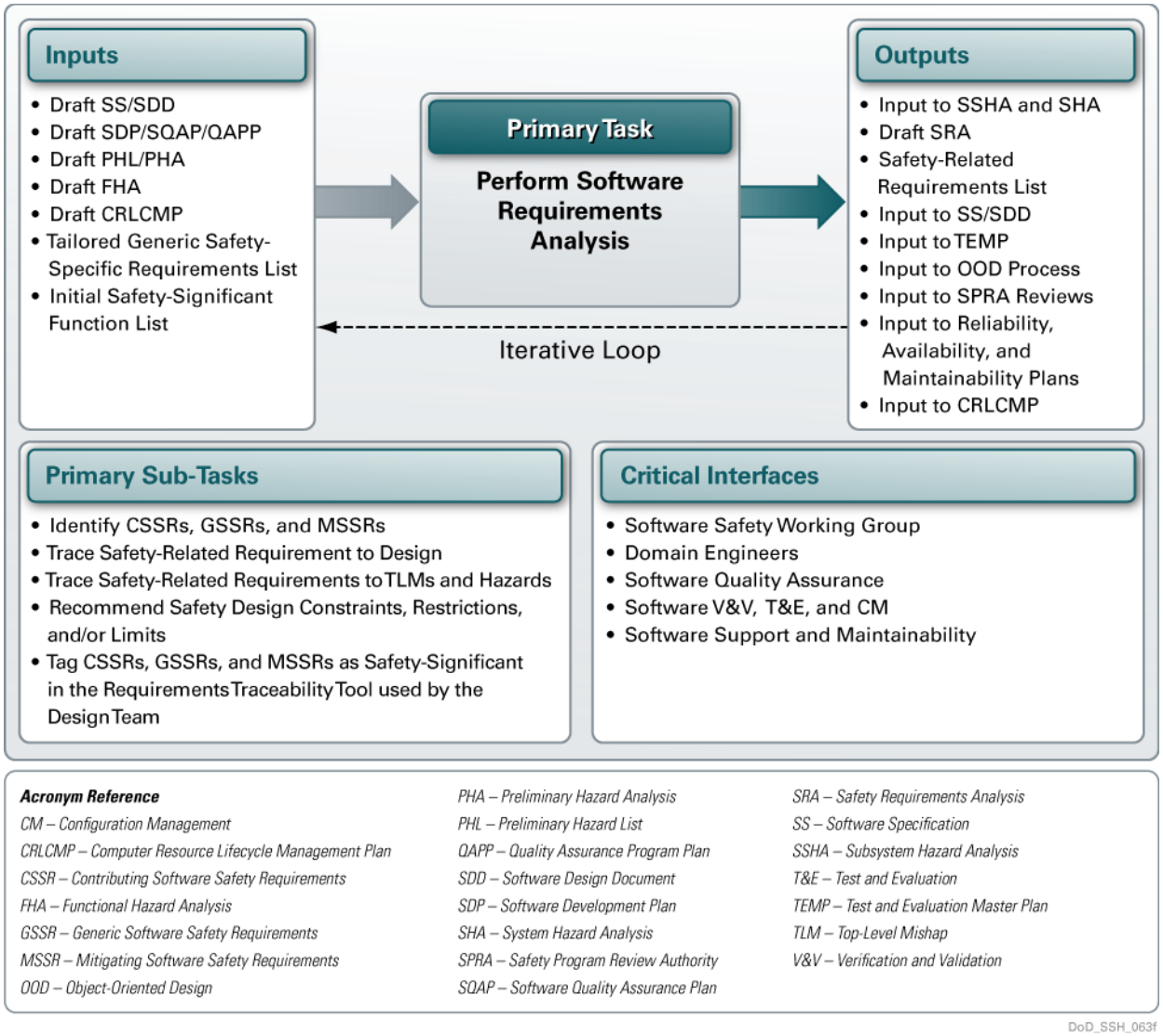

Figure 4-27: Safety Requirements Analysis

Safety requirement specifications identify the specifics and the decisions made based on the level of safety risk, desired level of safety assurance, and the visibility of software safety within the developer organization. Methods are dependent on the quality, breadth, and depth of initial hazard and failure mode analyses and on lessons learned from similar systems. The generic list of requirements and guidelines establishes the starting point which initiates the system-specific

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

SSR identification and implementation process. Identification of system-specific software requirements is the direct result of a complete hazard analysis methodology (see Figure 4-28).

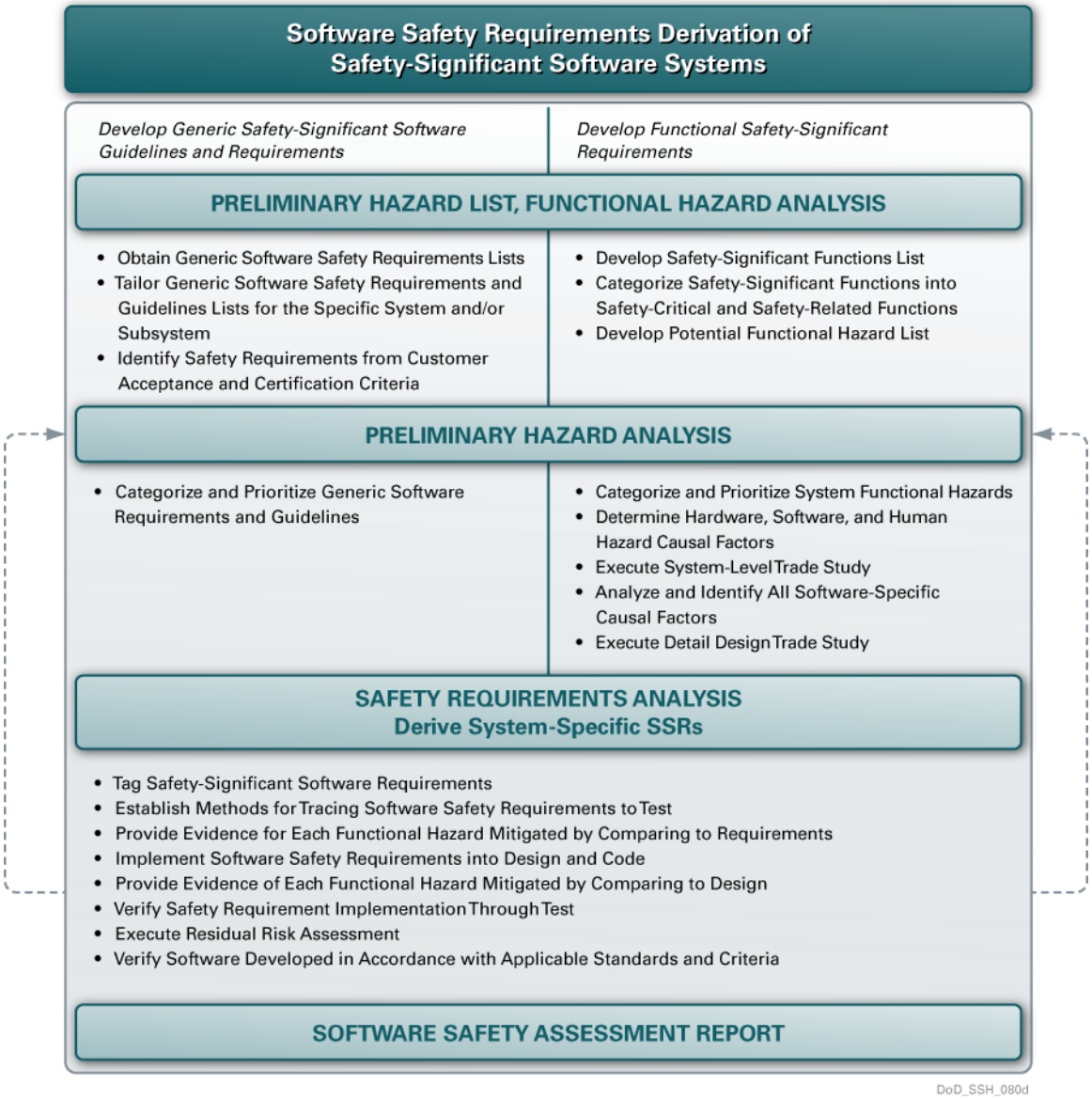

Figure 4-28: SSR Derivation

SSRs are derived from multiple sources, including generic lists, system functionality analysis, causal factor analysis, and the implementation of hazard controls. The analysis of system functionality identifies those functions in the system that, if not properly executed, can result in an identified system hazard. Correct operation of the function related to the SSRs is essential to the safety of the system. The software causal factor analysis identifies lower-level design requirements that, based on their relationship to safety-significant functions or in the context of

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

the failure pathway of the hazard, make them safety-significant as well. Design requirements developed to mitigate other system-level hazards (e.g., monitors on safety-critical functions in the hardware) are also SSRs.

The software safety engineer must present the SSRs to the customer (via the SwSSWG) for concurrence as to whether the SSRs eliminate or resolve the hazardous condition to acceptable levels of safety risk prior to implementation. The SSRs must possess a direct link between the requirement and a system-level hazard. The following paragraphs provide additional guidance on developing SSRs other than from generic lists. The LOR table must specify how each safety-significant requirement will be verified and validated.

4.3.5.1 Categories of Software Safety Requirements

The safety requirements of the system are analyzed and allocated to the design specifications from within the SRA process. Software safety requirements are those requirements that will be allocated specifically to the software design and documented in the SRS. The SRA process can further define these requirements into specific categories for safety and design management purposes. Designers must tag each of the SSRs in the appropriate requirements management tool and include the LOR necessary for design, implementation, and test tasks required for the integrity and assurance of safety functionality.

4.3.5.1.1 Contributing Software Safety Requirements

The CSSRs are requirements that should already exist in the specifications and were likely authored by someone other than a safety engineer. CSSRs are related to the performance of the system to accomplish its intended function or mission. These requirements are not present for the mitigation or control of a hazard; in fact, they will often contribute to the existence of a hazard. An example of a CSSR is “Fire the Weapon.” This is an essential requirement in the performance of the system. The system would not be complete in terms of operational capability without this requirement in the functional or software specification. This CSSR is a contributor to the existence of a hazard in the system. CSSRs are safety-significant functions and must be accounted for in the SRA for their contribution to the existence of a hazard or hazardous condition of the system.

4.3.5.1.2 Generic Software Safety Requirements

GSSRs are requirements that have been documented over the years under the heading of lessons learned and best practices. Examples of these requirements are located in documents such as North Atlantic Treaty Organization Standardization Agreement (STANAG) 4404 and Appendix E of this Handbook. GSSRs are usually authored by safety personnel or domain experts of specific systems. GSSRs are those design features, design constraints, development processes, best practices, coding standards and techniques, and other general requirements that are levied on a system containing safety-significant software, regardless of the functionality of the application. The requirements themselves are not safety specific and may not yet be tied to a specific system

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

hazard. GSSRs may also be identified as reliability requirements, good coding practices, etc. GSSRs are based on lessons learned from previous systems where failures or errors occurred that resulted in a mishap or potential mishap.

GSSRs that are defined and included in the SRA are those that are specifically considered applicable for the system being designed. If the specific task is executed correctly and completely, the relevant GSSRs introduced into the SRS will result in fewer MSSRs in later phases of the system acquisition. Detailed hazard analysis will discover hazard mitigations already in the design because of the implementation of a documented GSSR in the SRS.

Figure 4-29 depicts the software engineering process for tailoring the generic software safety requirement list. The PHL, PHA, and FHA will help determine the applicability of many individual generic requirements and how they should be implemented during preliminary design. The software safety analysis must identify the applicable GSSRs necessary to support the development of the SRS, as well as programmatic documents (e.g., Software Development Plan). A tailored list of these requirements must be provided to the software developer for inclusion into the SRS and other documents.

Several individuals, agencies, and institutions have published lists of generic safety requirements for consideration. Regardless of which list is used, the analyst must assess each item for applicability, compliance, or non-compliance. On a particular program, the agreed upon GSSRs should be included in the SRA and appropriate high-level system specifications.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

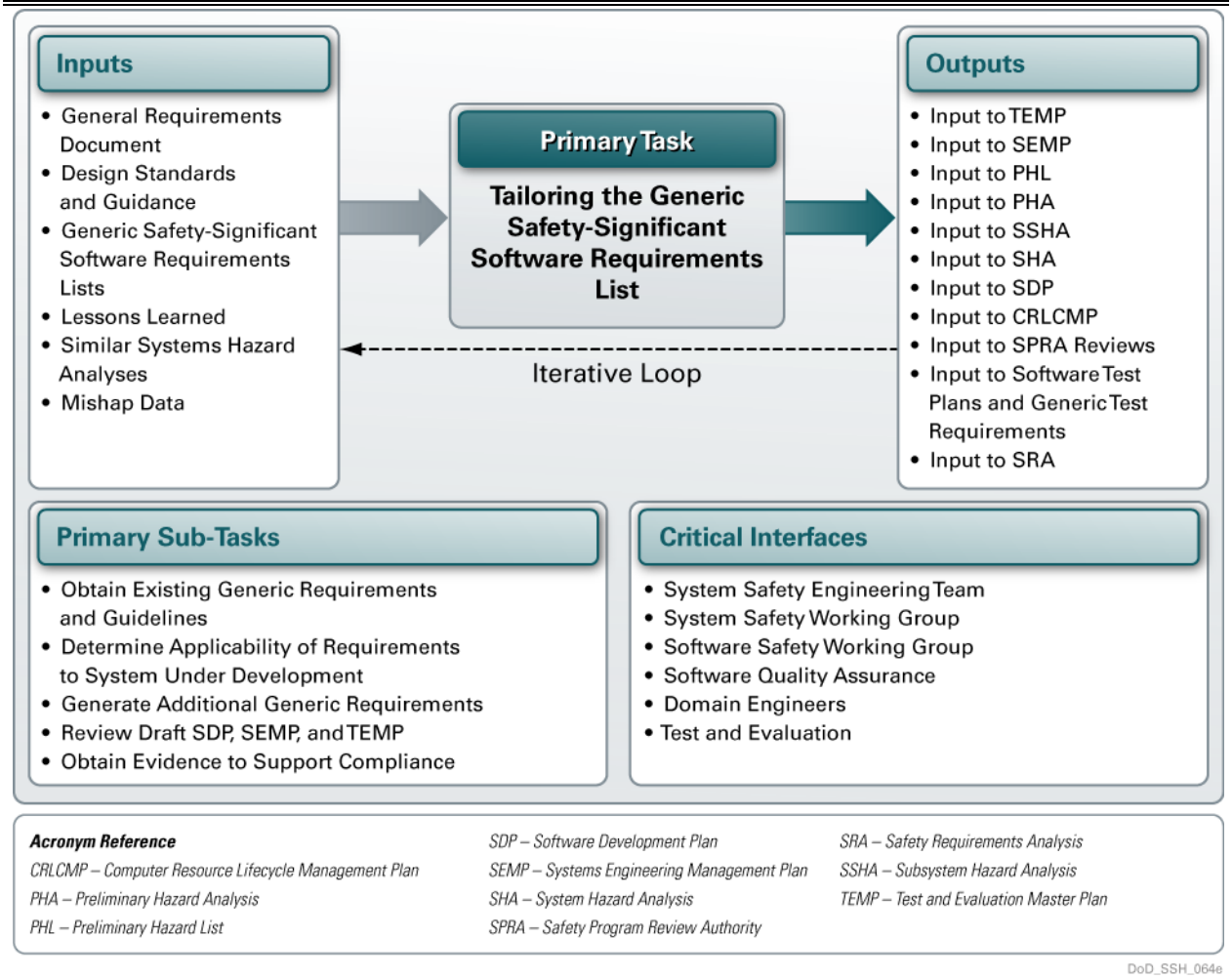

Figure 4-29: Tailoring the Generic Safety Requirements

Figure 4-30 is an example of a worksheet that may be used to track GSSR implementation. Guidelines can be established in the SDP and the design and coding guideline documents if the GSSRs have been established early in the acquisition process. The checklist worksheets can be developed for use during design and code development peer reviews to ensure that implementation evidences are adequately documented. Whether or not the program complies with the requirement, the physical location of the evidence of implementation must be cited in the "Evidence" block of the form. If the program does not comply with the requirement (e.g., too late in the development to impose a safety kernel) or the requirement is not-applicable (e.g., an Ada requirement when developing in C++ or JAVA), a statement of explanation must be included in the "Rationale" block. An alternative mitigation of the source risk that the requirement addresses should be described, if applicable, possibly pointing to another generic requirement on the list.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Generic Software Safety Requirements Implementation	Intended Compliance		
	Yes	No	N/A
Item			
Coding Requirements Issues Has an analysis (scaling, frequency response, time response, discontinuity, initialization, etc.) of the macros been performed?		X	
Rationale			
(If NO or N/A, describe the rationale for the decision and resulting risk) There are no macros in the design (discussed at checklist review 1/11/96)			
Evidence			
(If YES, describe the kind of evidence that will be provided. *Note: Specify sampling percentage per SwSPP, if applicable*)			
Action			
(State the Functional area with responsibility) Software Development POC:			

DoD_SSH_041b

Figure 4-30: Example Software Safety Requirements Tracking Worksheet

If using the "blanket" approach of establishing the entire list of guidelines or requirements for a program, note that each requirement will cost the program critical resources; personnel to assess and implement; budget for design, code, and testing activities; and program schedule. Unnecessary requirements will impact these factors and result in a more costly product with marginal benefit. Thus, these requirements should be assessed and prioritized according to applicability to the development effort. Inappropriate requirements which have not been adequately assessed are unacceptable. The analyst must assess each requirement individually and introduce only those that apply to the development program.

Some requirements necessitate a sampling of evidence to provide implementation (e.g., no conditional "GO-TO" statements, no "wait" statements, no unchecked conversions, etc.). The lead software developer will often gather the implementation evidence for the generic SSRs from those who can provide the evidence. The lead software developer may assign SQA, CM, V&V, human factors, software designers, or systems designers to fill out individual worksheets. The tailored list of completed forms should be approved by the system safety engineer, submitted to the SDL, and referred to by the SAR. This provides evidence of GSSR implementation.

4.3.5.1.3 Mitigating Software Safety Requirements

MSSRs are requirements derived from in-depth mishap and hazard causal analyses. As designs mature, the software development teams are implementing the best practice GSSRs that were

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

identified early in the development lifecycle. During this development process, the safety engineer is performing the safety analysis to determine whether the GSSRs have successfully mitigated the known causal factors of the mishaps and hazards. There will be instances where the GSSRs have not completely mitigated to acceptable levels of risk. At this point, MSSRs must be derived. These requirements mitigate or control mishap or hazard causes to acceptable levels of safety risk by accomplishing what the high-level GSSR failed to accomplish.

MSSRs are usually authored by safety engineers, with input and assistance from the design engineers and domain experts associated with the design or subsystem being analyzed. These MSSRs must be added to the specifications to ensure incorporation into the design and verification and validation for safety risk reduction and hazard closeout. MSSRs are usually identified late in the development lifecycle, and there will likely be considerable "push back" from the designers as "new requirements" are not embraced late in the schedule. Before authoring a new MSSR, the software safety engineer should first consider writing an "exception" or "deficiency" against an existing GSSR if a related GSSR exists in the specification.

4.3.5.2 Derive System Software Safety Requirements in the SRA

4.3.5.2.1 Preliminary SSRs

The initial attempt to identify system-specific SSRs evolves from the GSSRs identified from best practices and the FHA and PHA performed in the early phase of the development program. As previously discussed, PHL and PHA hazards are a product of the information reviewed pertaining to systems specifications, lessons learned, analyses from similar systems, common sense, and preliminary design activities. The analyst ties the identified hazards to functions in the system (e.g., inadvertent rocket motor ignition to the ARM and FIRE functions in the system software). The analyst flags these functions and associated design requirements as safety-significant and enters them into the Requirements Traceability Matrix within the SRA. The analyst should ensure that the system documentation contains appropriate safety requirements for these safety-significant functions (e.g., ensure that all safety interlocks are satisfied prior to issuing the ARM or FIRE command). Lower levels of specification will include specific safety interlock requirements satisfying the preliminary SSRs. These types of requirements are MSSRs.

The safety engineer also analyzes the hazards identified in the PHA and the safety-significant functions of the FHA to determine the potential contribution of the software. For example, a system design requires the operator to commit a missile to launch; however, the software provides the operator with a recommendation to fire the missile. This software is also safety-significant and must be designated as such and included in the RTM. Other safety-significant interactions may not be as obvious and will require more in-depth analysis of the system design. The analyst must also review the hazards identified in the PHA and develop preliminary design requirements to mitigate other hazards in the system. Many of these design requirements will include software, making that software safety critical. During the early design phases, the safety analyst identifies these requirements for consideration and inclusion in the SRS.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

These design requirements, along with the generic SSRs, represent the preliminary SSRs of the system, subsystems, and interfaces (if known). These preliminary SSRs must be accurately supported in the safety analyses and hazard tracking database for extraction when reporting the requirements to the design engineering team.

4.3.5.2.2 Matured SSRs

As the system and subsystem designs mature, the requirements unique to each subsystem also mature. During this phase of the program, the safety engineer attends design reviews and meetings with the subsystem designers to accurately define the subsystem hazards. The safety engineer documents the identified hazards in the hazard tracking database and identifies and analyzes the hazard causes. When using fault trees as the functional hazard analysis methodology, the causal factors leading to the root hazard help determine the derived safety-significant functional requirements. It is at this point in the design that preliminary design considerations are either formalized and defined into specific requirements or are eliminated if they no longer apply with the current design concepts.

The SSRs mature through analysis of the design architecture to connect the root hazard to the causal factor. The analyst continues the causal factors' analysis to the lowest level necessary for ease of mitigation (Figure 4-31). This helps mature the functional analysis commenced during preliminary SSR identification. The deeper into the design the analysis progresses, the more simplistic and cost effective the mitigation requirements tend to become. Additional SSRs may also be derived from the implementation of hazard controls (e.g., monitor functions, alerts to hazardous conditions outside of software, and unsafe system states). The PHA phase of the program should define causes to at least the computer software configuration item (CSCI) level. The SSHA and SHA should analyze causes to the algorithm level for areas designated as safety-significant.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

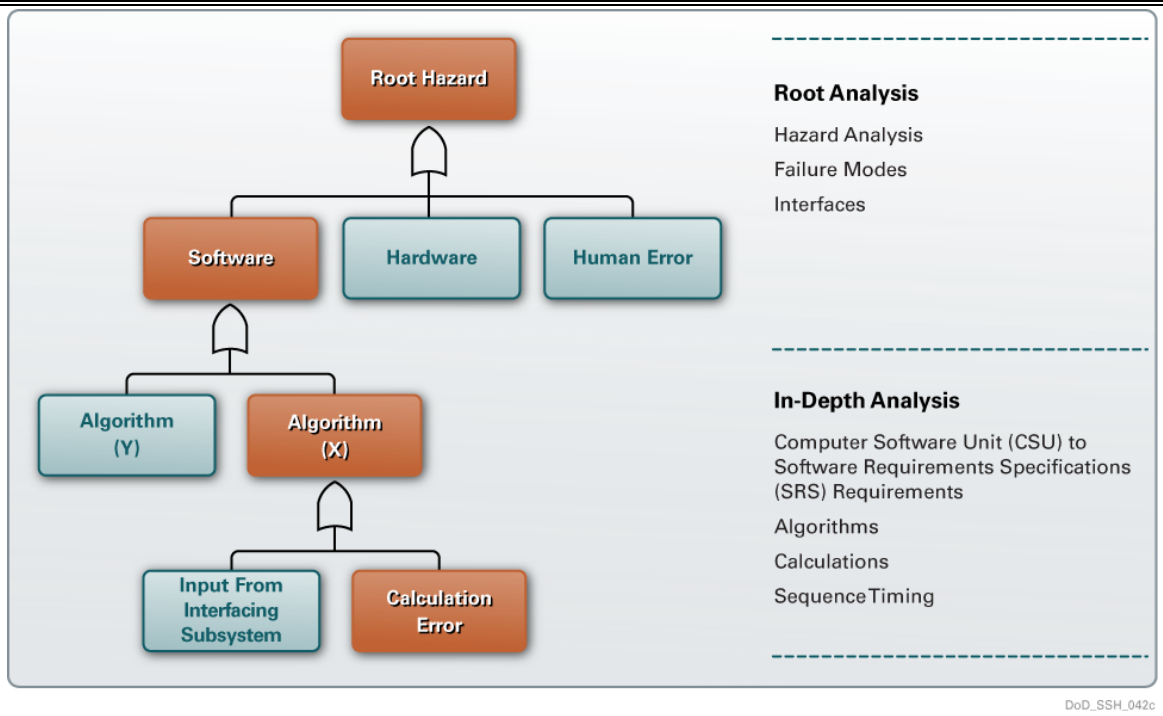

Figure 4-31: In-Depth Hazard Causal Analysis

The subsystem analysis begins during technology development and continues through the detailed design and CDR. The safety analyst must ensure that the safety analyses keep pace with the design. As the design team makes design decisions and defines implementations, the safety analyst must reevaluate and update the affected hazard records. In-depth analysis is considered complete when adequate mitigation of the root hazard is either evident in the design (evidence of AND gates in the fault tree to reduce the likelihood of failure propagation) or when safety mitigation requirements are defined by the safety and design or domain engineer (turning OR gates into AND gates in the fault tree).

4.3.5.3 Documenting SSRs

The SRA should document all identified SSRs. The objective of the SRA is to ensure that the intent of the SSRs in the system software is met and that the SSRs eliminate, mitigate, or control the identified causal factors. Mitigating or controlling the causal factors reduces the probability of hazards identified in the PHA. The SRA also provides the means for the safety engineer to trace each SSR from the system-level specification, to the design specifications, to software implementation, to individual test procedures and test results analysis.

The safety engineer uses this traceability, known as RTM, to verify that all SSRs can be traced from system-level specifications to design to test. The safety engineer should also identify all safety-significant SSRs to distinguish them as safety critical or safety related.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The RTM provides a useful tool for the software development group. The software group will be aware of the safety-critical and safety-related functions and requirements in the system. The group will also be alerted when making modifications to safety-critical CSCIs and computer software units (CSUs) that may impact SSRs. The SRA is a living document that the analyst constantly updates throughout the system development.

4.3.5.4 Software Analysis Folders

At this stage of the analysis process, it is good practice to begin the development of Software Analysis Folders. A SAF serves as a repository for all analysis data generated by the safety engineer on a particular CSCI. SAFs should be developed on a CSCI basis and should be made available to the entire SSS team during the software analysis process. Items to be included in the SAFs include, but are not limited to:

- Purpose and functionality of the CSCI source code listings annotated by the safety engineer
- Safety significant functions (both safety critical and safety related) and SSRs pertaining to the CSCI under analysis and SSR traceability results
- Design and development requirements established by the LOR assignment criteria
- Test procedures and test results pertaining to the CSCI
- Record and disposition of all Program Trouble Reports (PTRs) and Software Trouble Reports (STRs) generated against the particular CSCI
- A record of any and all changes made to the CSCI; SAFs need to be continuously updated during the preliminary and detailed design SSHA phases.

4.3.6 Preliminary Software Design, SSHA

At the preliminary design phase of the software development process, the software safety team will be participating in the implementation of specific LOR tasks identified in the LOR table. These tasks will be focused on determining how the software development team is interpreting and implementing the SSRs in the design architecture. This is an essential element of the total software safety effort. Complementary to the software assurance activities is the in-depth causal analysis to identify software's contribution to mishaps and hazards

The identification of subsystem and system hazards and failure modes inherent in the system under development (Figure 4-32) is also essential to the success of a credible software safety program. This method identifies the system hazards and failure modes and determines which hazards and failure modes are caused or influenced by software or lack of software. This determination includes scenarios where information produced by software could potentially influence the operator into a wrong decision, resulting in a hazardous condition (design-induced human error). Moving from hazards to software causal factors and design requirements to eliminate or control the hazard is practical, logical, and adds utility to the software development

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

process. This process is efficient because much of the analysis is accomplished to influence preliminary design activities.

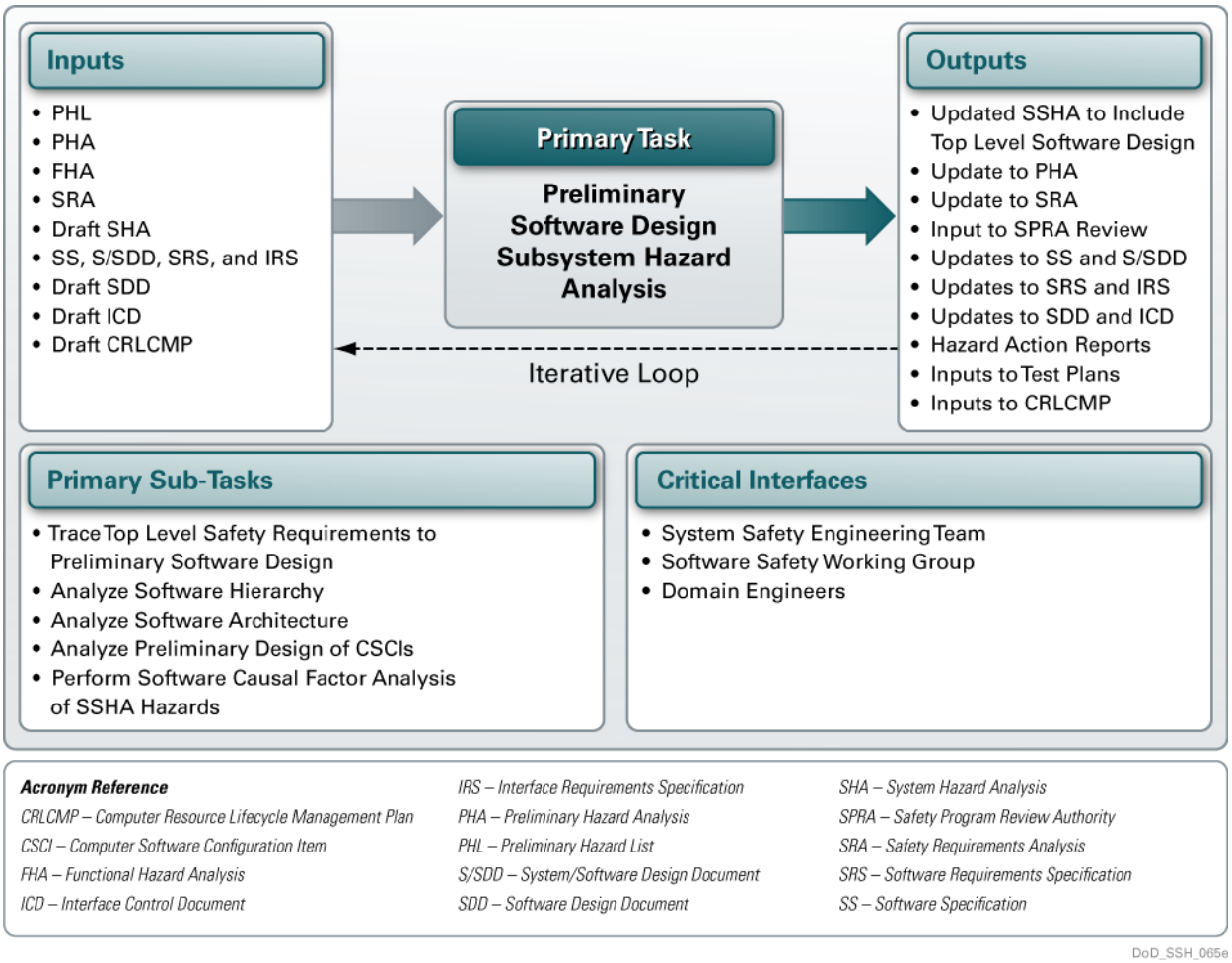

Figure 4-32: Preliminary Software Design Analysis

The specifics of performing the SSHA or SHA are described in Appendix C, Sections C.1.6 and C.1.7. MIL-STD-882C, Tasks 204 and 205, and other reference texts in Appendix B provide a more complete description of the process. The foundation of an SSP is a systematic and complete hazard analysis process.

One of the most helpful steps within a credible software safety program is to categorize the specific causes of the hazards and software inputs in each of the analyses (PHA, SSHA, SHA, and O&SHA). Hazard causes can be those from hardware (or hardware components), software inputs (or lack of software input), human error, software-influenced human error, or hardware or human errors propagating through the software. Hazards may result from one specific cause or any combination of causes. As an example, loss of thrust on an aircraft may have causal factors in all four categories. Potential causes include:

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Hardware – Foreign object ingestion
- Software – Software commands engine shutdown in the wrong operational scenario
- Human error – Pilot inadvertently commands engine shutdown
- Software-influenced pilot error – Computer provides incorrect, insufficient, or incomplete data to the pilot, causing the pilot to execute a shutdown.

For each identified cause, the safety engineer must identify and define hazard control considerations (PHA phase) and requirements (SSHA, SHA, and O&SHA phases) for the design and development engineers. The safety engineer communicates hardware-related causes to the appropriate hardware design engineers, software-significant causes to the software development and design team, and human error-related causes to the human factors organization or to the hardware or software design team. The safety engineer must also report all requirements and supporting rationale to the systems engineering team for evaluation, tracking, and disposition.

The preliminary software design SSHA begins with the identification of the software subsystem and uses the derived system-specific SSRs. The purpose is to analyze the system and software architecture and preliminary CSCI design. At this point, the analyst has (or should have) identified all SSRs (e.g., SSRs, generics, functional-derived requirements, and hazard control requirements) and begins allocating them to the identified safety-related or safety-critical functions and tracing them to the design.

The allocation of SSRs to the identified hazards can be accomplished through the development of SSR verification trees (Figure 4-33) which link safety-critical and safety-related SSRs to each SCF. The SCFs are already identified and linked to each hazard. By ensuring that the SCF has been safely implemented, the hazard will be controlled. The tree allows the software safety engineer to verify that controls have been designed into the system to eliminate, control, or mitigate the SCF.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

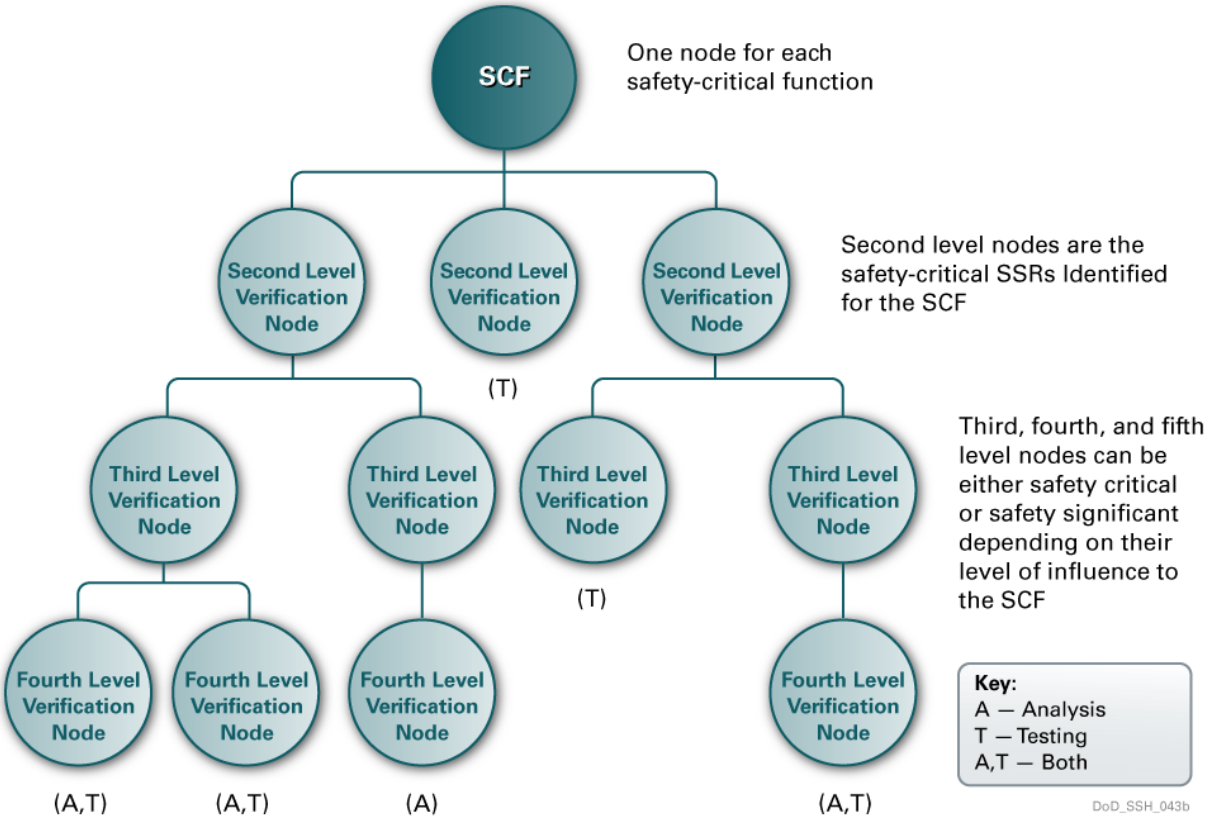

Figure 4-33: SSR Verification Tree

The root node of the tree represents one SCF. The safety analyst needs to develop a verification tree for each system-level SCF. The second level nodes are the safety-critical SSRs linked to each system-level SCF. The third, fourth, and lower level nodes represent the safety-critical or safety-related SSRs allocated to each SCF. The fourth- and fifth-level nodes are developed to fulfill the level of detail required by the SSS team. By verifying the nodes through analysis and testing, the safety analyst verifies the correct design implementation of the requirements.

The choice of analysis and/or testing (including demonstrations) to verify the SSRs is determined by the LOR criteria defined early in the development lifecycle, which is based on the criticality of the requirement to the overall safety of the system and the command and control capability of the function. Whenever possible, the safety engineer should use testing for verification. The safety engineer can develop an SSR Verification Matrix, similar to Table 4-7, to track the verification of each SSR or directly document the verification in the RTM. The choice is dependent on the size and complexity of the system. If developed, the SSR matrix should be included as an appendix to the SRA, and the data should feed directly into the RTM. The safety analyst should also update the hazard tracking database and safety application functions with the analysis and test results once verification is complete.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table 4-7: Example of an SSR Verification Matrix

SSR	Test/Analysis	Verified (Date)	Comments
1	Test: (TP4-1 and TP72-1) Analysis: (CSCI/CSU Name)	7/14/10	**Test Passed.** Test Data found on Data Extract SharePoint Site #10
1.2	Test: (TP2-2)	9/1/10	**Test Failed.** STR/PTR JDB002 generated and submitted to design team
1.3	Analysis: (CSCI/CSU Name)	9/23/10	Analysis of CSCI/CSU *(Name)* indicated successful implementation of the algorithm identified by SSR 1.3

DoD_SSH_044b

The next step of the preliminary design analysis is to trace the identified SSRs and causal factors to the design (to the actual CSCIs and CSUs). The RTM is the easiest tool to accomplish this task (Table 4-8). Other methods of preliminary design hazard analysis include the Module Safety-Criticality Analysis and Program Structure Analysis, which are discussed in Sections 4.3.6.3.1 and 4.3.6.3.2, respectively.

Table 4-8: Example of an RTM

SSR	Requirement Description	CSCI	CSU	Test Procedure	Test Results

DoD_SSH_045

4.3.6.1 Attributes of Fault Management

During the safety analysis of the preliminary software design where the FHA, PHA, and SRA are completed to the first order of design maturity, the safety engineer and software designer must define specific requirements for addressing software faults. In a safety-critical system, the safety engineer defines specific requirements to reduce the likelihood of mishap or failure. It is imperative that the designer defines the design attributes of the system to detect, manage, and

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

take corrective action when failure occurs. Specific design fault attributes that must be considered are:

- Fault Detection – Detecting specific faults that could lead to or contribute to Catastrophic or Critical mishaps or hazards
- Fault Isolation – Isolating specific faults and failure conditions to preclude them from contributing to or propagating Catastrophic or Critical mishaps and hazards
- Fault Annunciation – Annunciating the fault or failure condition to a human operator for understanding of the impending hazardous condition within the system and allowing them to take the appropriate (and defined) corrective action. In some cases, the annunciation may be between software systems, one being the actor and the other being the monitor and mitigation. In each case, the same annunciation considerations of protocol, timing, content, and tagging need to be addressed for a complete solution
- Fault Tolerance – Tolerating specific (and defined) faults or failure conditions (after detection, isolation, and annunciation) and transitioning the system to a known safe or operational mode or state
- Fault Recovery – Specific design features to take corrective action from faults or failure conditions to allow the system to remain in a safe and operational state.

Most of these requirements are classified as MSSRs and must be adequately identified and analyzed in the SRA and formally documented in the SRS.

4.3.6.2 Other Design Considerations for Safety

During the SRA and preliminary design analysis processes, the safety engineer and software designer must consider the specific implementation of many of the GSSRs documented in the SRS during the tailoring of generic safety requirements. This includes the attributes of fault management and the functional and physical partitioning of the design.

Functional partitioning refers to the modularity of the design to ensure that safety-significant functionality (especially for safety-critical functions) is partitioned separately from non-safety-significant functionality. This effort reduces the likelihood of faults or failures of non-safety-significant functionality contributing to or influencing safety-significant functions that would lead to hazardous conditions or mishaps. Safety-significant functionality should minimize complexity to reduce the likelihood of failure from functionality that contributes minimally to the basic design baseline of the safety-significant function. Simple safety-critical functions are easier to design, implement, and test to verify integrity than those that are more complex. These efforts will also have an effect on software maintenance and upgrades. With functional partitioning, the potential for negative effects on the safety of the software is reduced. Without associated data partitioning, the fault may cross a functional partition through contaminated data. This will force the analyst to check functional, data, and timing partitions together.

Physical partitioning applies to designs that possess more than one processor. On many complex safety-critical systems, safety functionality may only be allowed to execute on a specific

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

processor, while non-safety-significant functionality would be required to execute on the other. Likewise, running on separated channels or lanes physically separates the function, but these functions eventually merge into a decision point that must be checked. This forces the analyst to check and verify the "edges" or boundaries of any partition.

4.3.6.3 Example Techniques of Preliminary Software Design Analysis

Numerous techniques can be employed by the software safety team to fulfill the criteria for performing the preliminary software design analysis. The technique selected should be based on the value of the analysis technique in producing the results desired by the team or required by the contract. In selecting the proper technique, the analyst must always consider the essential information required by the analysis. Specific questions that should be considered are:

- What is required by the SOW, SSMP/SSPP, or the contract?
- What do I hope to obtain by performing the analysis?
- What analysis techniques am I most proficient in?
- Is there measurable value added by performing the analysis?
- Will the product produced be useful to the design team, test team, or safety team?
- What engineering evidences or artifacts will best fulfill the acceptance or certification criteria of the system?
- What metrics are being tracked by the program that include or can be used for safety?

4.3.6.3.1 Module Safety-Criticality Analysis

Module (CSCI or CSU) safety-criticality analysis determines which CSCIs or CSUs are safety-critical to the system in order to assist the safety engineer in prioritizing the level of analysis or LOR to be performed on each module. The safety analyst bases the priority on the degree at which each CSCI or CSU implements a specific safety-significant function. The analyst can develop a matrix (Table 4-9) to illustrate the relationship each CSCI or CSU has with the safety-significant functions. The matrix should include all CSCIs and CSUs required to perform a safety-significant function, such as math library routines which perform calculations on safety-significant data items. The criticality matrix should list each routine and indicate which safety-significant functions are implemented. Symbols can be used to note importance with respect to accomplishing a safety-significant function.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Table 4-9: Example Safety-Significant Function Matrix

Safety-Significant Functions							
CSCI/CSU Name	SSF 1	SSF 2	SSF 3	SSF 4	SSF 5	SSF 6	Rating
INIT	M		M	M			M
SIGNAL		H	M				H
D1HZ					H		H
CLEAR				H		H	H
BYTE							L

DoD_SSH_046a

Acronym Reference

H – High Risk *M – Medium Risk*
L – Low Risk *SSF – Safety-Significant Function*

The last column in the matrix is the overall criticality rating of the CSCI or CSU. The analyst should place an H, M, or N in this column based on the highest level of criticality for that routine. However, if a CSCI or CSU has a Medium criticality over a number of SCFs, the cumulative rating may increase to the next level. Much of the information required as input for this analysis can be obtained from the FHA and the LOR assessment.

4.3.6.3.2 Program Structure Analysis

Regardless of the programming language chosen, a program uses a hierarchical decomposition in its design. In general, a program is made up of computer software configuration items, computer software components (CSCs), and computer software units. CSCIs are usually responsible for a single top-level function or major division of the overall program. CSCIs are made up of two or more CSCs, which further break down into lower level CSCs or CSUs.

The purpose of program structure analysis is to reconstruct the program hierarchy (architecture) and overlay structure, and to determine if any safety-significant errors or concerns exist in the structure. The program hierarchy should be reconstructed on a CSCI-level in the form of a control tree. The system safety engineer begins by identifying the highest CSU and its call to other CSUs. The system safety engineer performs this process for each level of CSUs. When the control flow is complete, the safety engineer identifies recursive calls, extraneous CSUs,

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

inappropriate levels of calls, discrepancies within the design, calls to the system and library CSUs, calls to other CSCIs, and CSUs not called. CSUs called by more than one name and units with more than one entry point are also identified. Figure 4-34 provides an example hierarchy tree.

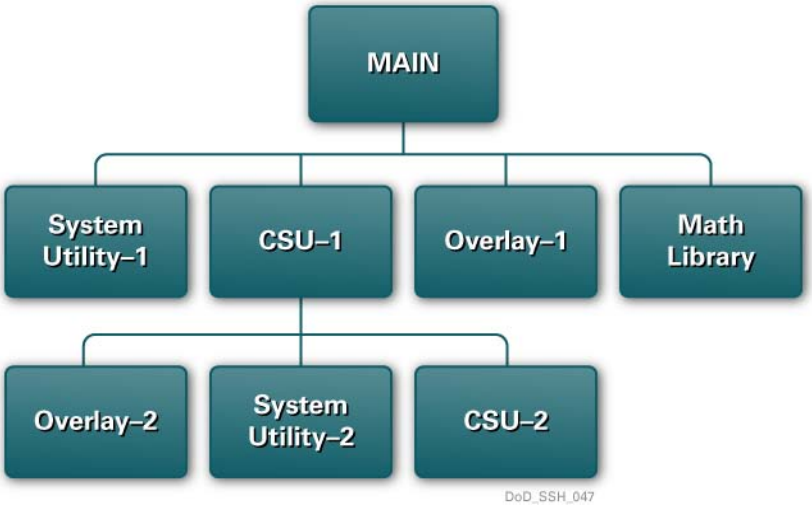

Figure 4-34: Hierarchy Tree Example

All existing overlays and patches should be identified and eliminated due to the added potential risk they can introduce. If overlays or patches are allowed to remain, the following issues should be considered:

- Overlaying and patching should not be unnecessarily performed
- Safety-critical code should not be in overlays or be patched
- All overlay loads and patch applications should be verified and proper actions taken if an overlay or patch cannot be loaded (in some cases, the system will halt; in others, some recovery is sufficient, depending on the criticality and impact of the failure)
- Note the effect that the overlay structure and patches have on the time-critical code
- Interrupts are enabled when an overlay or patch is loaded
- Review which overlays and patches are loaded at all times to determine if they should be incorporated into resident code to cut down on system overhead
- Overlays and patches must comply with the guidelines and criteria of governing, certification, or contractual standards and comply with all verification and testing criteria to certify the system configuration.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.3.6.3.3 Traceability Analysis

The analyst develops and analyzes the RTM to identify where SSRs are implemented in the code, SSRs that are not being implemented, and code that does not fulfill the intent of the SSRs. The traced SSRs should include those identified by top-level specifications and those identified by the SRS, Software Design Document, and Interface Control Document/Interface Design Specification (IDS). This trace provides the basis for analysis and test planning by identifying the SSRs associated with the code. This analysis traces SSRs from specifications to design and test and identifies safety-critical items.

Tracing encompasses a requirement-to-code trace and a code-to-requirement trace. The forward trace, requirement-to-code, identifies the requirements that belong to the functional area (if not already identified through requirement analysis). The forward trace then locates the code implementation for each requirement. A requirement may be implemented in more than one place, making a matrix format very useful.

The backward trace, code-to-requirement, is performed by identifying the code that does not support a requirement or a necessary "housekeeping" function. In other words, the code is extraneous (e.g., debugging code left over from the software development process). The safety analyst performs this trace through an audit of applicable code after they have a good understanding of the corresponding requirements and system processing. Code that is not traceable should be documented and eliminated if practical. This effort also supports the testing effort because it may identify requirements that still need to be tested.

The following items should be documented for this activity:

- Requirement-to-code trace
- Unit(s) (code) implementing each requirement
- Requirements that are not implemented
- Requirements that are incompletely implemented
- Code-to-requirement trace
- Unit(s) (code) that is not directly or indirectly traceable to requirements or necessary "housekeeping" functions.

4.3.7 Detailed Software Design, SSHA

Detailed design-level analysis (Figure 4-35) follows the preliminary design process where the SSRs were traced to the CSCI level and is initiated after the completion of the PDR. Prior to performing this process, the safety engineer should have completed development of the fault trees for all top-level hazards, identifying all potential causal factors and deriving generic and functional SSRs for each causal factor. During this process, the system safety engineer works closely with the software developer and IV&V engineers to ensure that the SSRs have been

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

implemented as intended and that this implementation has not introduced additional safety concerns.

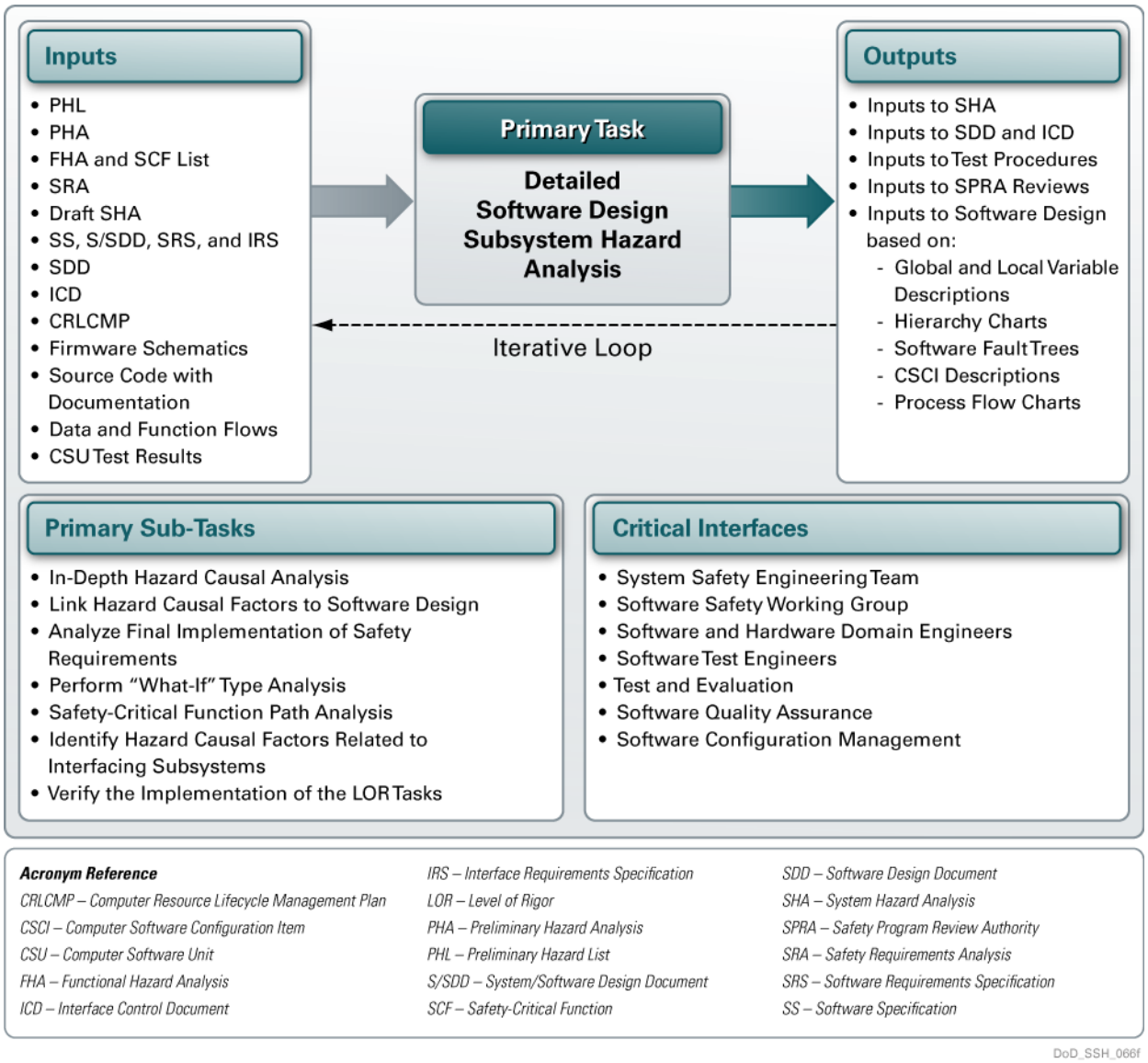

Figure 4-35: Detailed Software Design Analysis

4.3.7.1 Participate in Software Design Maturation

Detailed design analysis provides the SSS engineer and software development experts with an opportunity to analyze the implementation of SSRs at the CSU level. At the CSU level, the verification method usually entails inspection by the original designer and peer review. Detailed design analysis takes the software safety engineer from the CSCI level one step further into the computer software architecture.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

As the software development process progresses from preliminary design to detailed design and code, the safety engineer is responsible for communicating the intent of the SSRs to the appropriate engineers and programmers on the software development team. In addition, the safety engineer must monitor the software design process to ensure that the software engineers are implementing the requirements into the architectural design concepts. This is accomplished through interactive communication between safety engineering and software engineering. It is essential that the software developer and the software safety engineer work together in the inspection, analysis, and verification of the SSRs.

In today's software environment, the software safety engineer cannot be expected to be an expert in all computer languages and architectures. Software design reviews, peer reviews, code walkthroughs, and technical interchange meetings will help provide a conduit of information for the assessment of the software development program from a safety perspective. This assessment includes how well the software design and programming team understands the system hazards and failure modes attributed to software inputs or influences. The assessment also includes willingness to assist in the derivation of software safety requirements, the ability to implement the requirements, and the ability to derive test cases and procedures to verify the resolution of the safety hazard.

Most programs are resource limited, including most system safety engineering support functions and disciplines. There will not be sufficient time or resources for the safety team to attend every design meeting. The safety manager is responsible for prioritizing meeting attendance and reviews which have the most value added to the safety resolution function. This prioritization is dependent on the amount of communication and trust between disciplines and among team members.

It is important to remember that there is a link between the SSRs and causal factors identified by the fault tree analysis during the PHA and SHA phase of the software safety process. There are three methods of verifying SSRs—analysis, inspection, testing, or a combination of all three—as illustrated in Figure 4-36. Recommended approaches and techniques for analysis will be discussed in the subsequent paragraphs, while approaches for SSR verification through testing will be discussed in Section 4.4.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

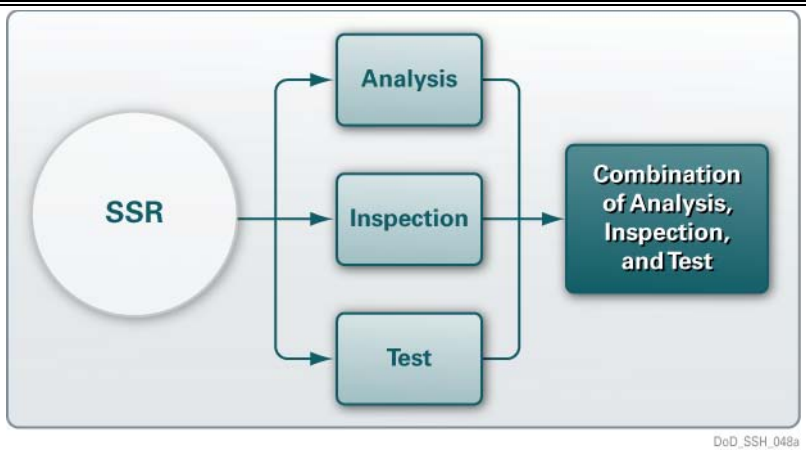

Figure 4-36: Verification Methods

4.3.7.2 Detailed Design Software Safety Analysis

One of the primary analyses performed during detailed design analysis is to identify the CSU where an SSR is implemented. The term CSU refers to the code-level routine, function, or module. The software safety engineer will review with the software developer, IV&V engineer, or quality assurance engineer and begin to tag or link individual SSRs to CSUs, as illustrated in Figure 4-37. This helps focus the software safety engineer on the safety-significant processing, which is more intensive for large-scale development projects than for smaller, less complex programs. This analysis provides an opportunity to continue development of the RTM and allows the safety engineer to verify that the software development tasks are being accomplished in accordance with the software assurance LOR table defined early in the program.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

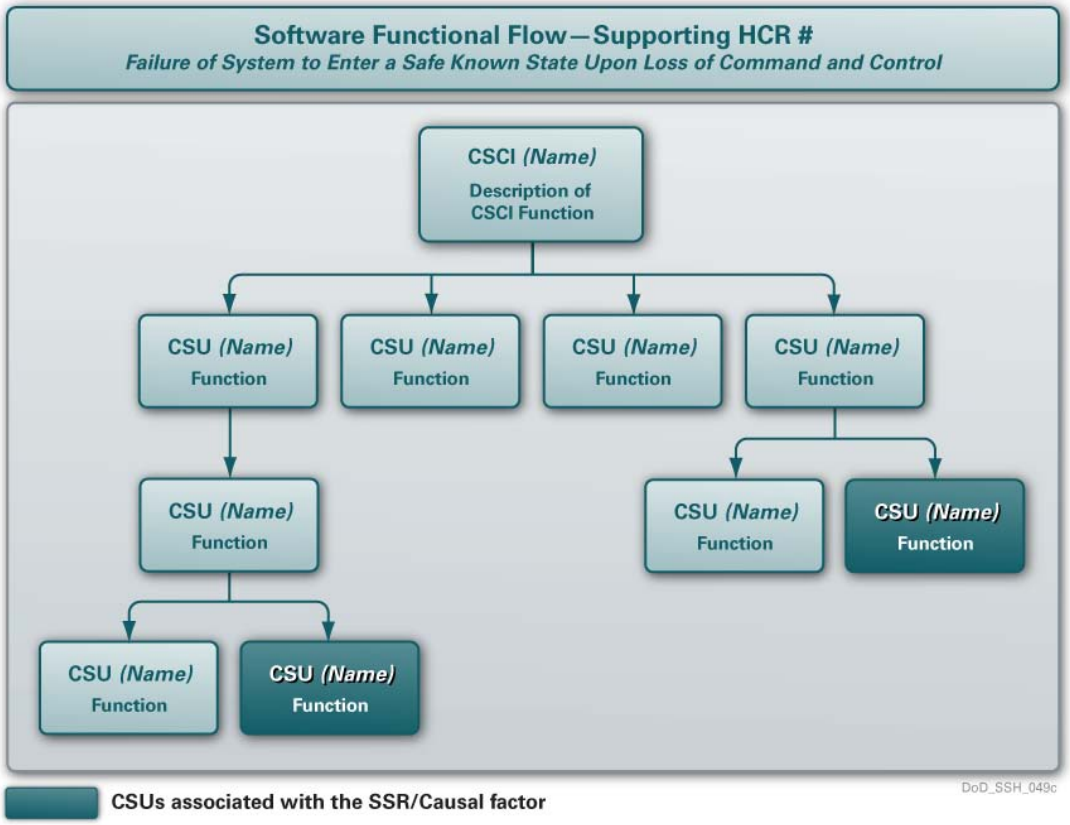

Figure 4-37: Identification of Safety-Significant CSUs

As previously discussed, Table 4-8 illustrates an example of an RTM template. The RTM contains multiple columns, with the left-most column containing the list of SSRs. Adjacent to this column is a description of the SSR and the name of the CSCI where the individual SSR has been implemented. The next column contains the name of the CSU where the SSR has been implemented. The next column contains the name of the test procedure that verifies the implementation of the SSR. The last column documents test results with pertinent comments. In some cases, it may only be possible to tag or link the SSR to the CSCI level because of the algorithms employed or the implementation of the SSR. In this case, the SSR will likely be verified through an extensive analysis effort rather than testing. The RTM should be included as part of the SRA report to provide evidence that all safety requirements have been identified and traced to the design and test.

After the RTM has been populated and all SSRs have been tagged or linked to the application code, the individual CSUs will be analyzed to ensure that the intent of the SSR has been satisfied. The appropriate developers and engineers should consult on this task. Process flow charts and DFDs are examples of tools that can aid in this process. These tools can help the engineer review and analyze software safety interlocks such as checks, flags, and firewalls that have been implemented in the design. Process flows and DFDs are also useful in performing "What If" and safety-critical path analyses and identifying potential hazards related to interfacing

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

systems. The safety application functions should include the products resulting from these safety tasks

4.3.7.2.1 Safety Interlocks

Safety interlocks can be either hardware or software oriented. As an example, a hardware safety interlock would be a key switch that controls a safe/arm switch. Software interlocks generally require the presence of two or more software signals from independent sources to implement a particular function. Examples of software interlocks are checks and flags, firewalls, come-from programming techniques, and bit combinations.

4.3.7.2.1.1 Checks and Flags

Checks and flags can be analyzed by reviewing the variables used by a CSU, ensuring that the variable types are declared and used accurately from CSU to CSU, and ensuring that the variables have been logically implemented. A life-threatening example is a hospital that uses computers to administer medication to patients. Within this system, there is one CSU that sets a flag every six hours that signals the machine to administer medication. However, if the flag uses an incorrect timer routine (logic error), the machine may set the flag every hour, resulting in an overdose of medication to the patient.

In general, safety functions must exhibit strong data typing. As an example, functions should not employ the logic of "1" and "0" to denote the safe and armed (potentially hazardous) states. The armed and safe states should be represented by at least a unique four-bit pattern and the safe state should be a pattern that cannot (as a result of a one, two, or three bit error) represent the armed pattern. The armed pattern should also not be the inverse of the safe pattern. If a pattern other than these two unique codes is detected, the software should flag the error, revert to a safe state, and notify the operator if appropriate.

Compilers need to be confirmed to not use the same flag for all arguments or checks. For example, if all IF statements and all other (“ANDing”, “less than,” “greater than”) branching and comparison (“EQUAL to”) statements use the same central processing unit (CPU) flag after compilation, the entire system is subject to a single point failure via that CPU flag. This has been found in several compilers, which means that it does not matter how many safety checks and interlocks are in the system if they all go true at once with one failure.

4.3.7.2.1.2 Firewalls

To isolate one area of software processing from another, software developers use firewalls. Firewalls are a type of partition used to isolate safety-critical processing from non-safety-critical processing. For example, assume that there is a series of signals that must be present in order for medication to be administered. The actual administration of the medication would be considered as safety-critical processing, while the general processing of preparing the series of signals would be non-critical. However, it takes the combination of all of the signals to activate

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

administration of the medication. The absence of any one of these signals would inhibit the medication from being administered. Therefore, it would take multiple failures to cause the catastrophic event. In the checks and flags example, this type of safety interlock would have prevented the failure of one event causing an overdose.

4.3.7.2.1.3 Come-From Programming

Come-from programming protects or isolates safety-critical code from non-safety-critical code. Come-from programming is rigorous to implement, and there are few compilers available on the market to support this technique. The difference in this technique is that it requires the application processing to know where it is at all times by using a program counter (PC) and to know where it has been (i.e., where it has "come from"). By knowing where it is and where the previous processing has been, the application can make validity checks to determine if the processing has stepped outside of its intended bounds.

For the medication example, the safety-critical processing CSU "ADMIN" will only accept an "administer dose" request from CSU "GO." The "ADMIN" CSU would then perform a validity check on the origin of the request. An answer other than "GO" would result in a "reject", and "ADMIN" would either ignore the request or perform error processing. This type of processing also prevents inadvertent jumps from initiating safety-critical functions. In this example, an inadvertent jump into the "ADMIN" routine would compare the value of the new PC to the value of the previous PC. Having preprogrammed "ADMIN" to only accept the PC from the "GO" CSU, "ADMIN" would recognize the error and perform the appropriate error processing. While this may appear expensive, classified systems and security programs require this type of rigor.

4.3.7.2.1.4 Bit Combinations

Bit combinations are another example of implementing safety interlocks in software. Bit combinations allow the programmer to concatenate two or more variables together to produce one variable. This one variable would be the safety-critical variable or signal, which would not be possible without the exact combination of bits present in the two variables that were concatenated together.

4.3.7.2.2 "What If" Analysis

"What If" analyses are used to speculate how certain processing will react given a set of conditions. These analyses allow the system safety engineer to determine if all possible combinations of events have occurred and to test how these combinations would react under credible and non-credible events. For example, how would the system react to power fluctuations or interrupts during processing? Would the state of the system be maintained? Would processing restart at the interrupting PC + 1? Would all variables and data be corrupted? These questions need to be asked of code that is performing safety-critical processing to ensure that the programmer has accounted for these scenarios and system environments. A "What If" analysis should also be performed on all "IF", "CASE", and "CONDITIONAL" statements used

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

in safety-critical code to ensure that all possible combinations and code paths have been accounted for and to avoid any extraneous or undesired code execution. In addition, this process allows the analyst to verify that there are no fragmented "IF", "CASE", or "CONDITIONAL" statements and that the code has been programmed top-down and is properly structured.

4.3.7.2.3 Safety-Critical Path Analysis, Thread Analysis, and UML Sequence Diagrams

The system safety engineer uses safety-critical path analysis to review and identify all possible processing paths or threads within the software and to identify which paths are safety critical, based on the identified system-level hazards and the predetermined safety-critical functions of the system. In this case, a path would be defined as events that, when performed in a series (one after the other), cause the software to perform a particular function. The UML sequence diagram adds some rigor in that timing and the originator or handler of the thread is identified along the way. Safety-critical path analyses use the identified system-level hazards to determine whether or not a particular function is safety-critical or not safety-critical. Functional Flow Diagrams (FFDs) and DFDs are excellent tools for identifying safety-critical processing paths and functions. In most cases, these diagrams can be obtained from the software developers or the IV&V team to save cost and schedule of redevelopment.

4.3.7.2.4 Identifying Potential Hazard Causes Related to Interfacing Systems

In SoS environments, it is necessary to analyze the functionality of the system and how it could contribute to mishaps in other systems that are functionally or physically connected. Conversely, it is important to understand how those other systems can influence the system. Detailed design analysis allows the system safety engineer to identify potential hazards related to interfacing systems. This is accomplished through interface analysis, SoS hazard analysis, models, and simulations at the IDS/Interface Control Document level. Erroneous safety-critical data transfers between system-level interfaces can be a contributing factor (causal factor) to a hazardous event.

These analyses should include an identification of all safety-critical data variables and timing, while ensuring that strong data typing has been implemented for all variables deemed safety critical. The interface analysis should also include a review of the error processing associated with interface message traffic and the identification of any potential failure modes that could result if the interface fails or if the transferred data is erroneous. Identified failure modes should be tied or linked back to the identified system-level hazards. The hazards should also be tagged to identify the stakeholder and be tied to the systems engineering risk management tools so that risk acceptance can be closed. This will ensure that the stakeholder is aware of all safety risk for which they will take ownership responsibility.

The most challenging aspect of this type of interface analysis is the identification of hazardous contributions of the system within the SoS environment that are not hazardous within its own boundaries. It is imperative that joint SoS working groups define the functional interfaces between systems in the SoS environment that are considered hazardous and that have the potential to lead to mishaps. Failure to do so can result in delays during testing and at risk acceptance or fielding.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.3.7.3 Detailed Design Analysis Related Sub-Processes

4.3.7.3.1 Process Flow Diagram Development

Process Flow Diagram (PFD) development is a line-by-line regeneration of the code into flow chart form. PFDs can be developed using a standard IBM flow chart template or by freehand drawing. PFDs provide the link between the FFD and DFD and allow the system safety engineer to review processing of the entire system in a step-by-step logical sequence. PFD development is time consuming and costly, which is one of the reasons it is a sub-process of Detailed Design Analysis. If the diagrams can be obtained from the software developers or IV&V team, the PFDs can be useful; however, the benefits of reverse engineering the design into PFDs do not provide a lot of value to the safety effort in performing hazard causal factor analysis. The real value of PFD development lies in the V&V that the system is performing the way it was designed to perform. From a system safety perspective, the primary benefit of process flow chart development is that it allows the system safety engineer to develop a thorough understanding of the system processing, which is essential when performing hazard identification and causal factor analysis. A secondary benefit is that by reverse engineering the coded program into flow chart form, the system safety engineer can verify that the software safety interlocks and safety-critical functionality have been implemented correctly and as intended by the top-level design specifications.

4.3.7.3.2 Code-Level Analysis

Code-level analysis is an activity that must be specifically called out in the LOR table for code that implements safety-critical functionality. Historically, non-nuclear DoD programs failed to call for code-level analysis due to the time, cost, and resources required to conduct the analysis. This analysis is essential to identify those code areas that convert, modify, or use safety-significant data for decision making. In addition, the code-level analysis should focus on how the developer has implemented the safety-significant code functions. The analysis should ensure that the developer has not introduced new hazard conditions or defeated existing safety features required by the system. The key for the analyst is to focus on where safety-significant data is converted, modified, created, and used to make safety-significant decisions.

A variety of techniques and tools may be applied to the analysis of code, depending on the programming language, criticality of the software, and resources available to the software safety program. The most common method is analysis by inspection. Use of structured analysis methodologies such as FTA, Petri Nets, data and control flow analyses, and formal methods is common at all levels of design and complexity. None of the techniques are comprehensive enough to be applied in every situation and are often used together to complement the others.

Code-level analysis begins with an analysis of the architecture to determine the flow of the program, calls made by the executive routine, the structure of the modules, the logic flow of each module, and the implementation in the code. Regardless of the technique used to analyze the code, the analyst must first understand the structure of the software, how it interacts with the system, and how it interacts with other software modules.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.3.7.3.2.1 Data Structure Analysis

The purpose of data structure analysis is to verify the consistency and accuracy of the data items used by a particular program. This analysis determines how the data items are defined and ensures this definition is used consistently throughout the code. A table (Table 4-10) can be constructed consisting of all data items used. The table should contain the name of the data item, the data type (integer, real, or Boolean), the variable dimension (16, 32, or 64 bit), the names of routines accessing the data item, whether the data item is local or global, and the name of the common block (if used). The appropriate SAF should contain the product developed from these safety tasks.

Table 4-10: Data Item Example

Data Item Name	Data Type	Data Dimension	Routine Accessing	Global/Local	Common Block
JINCOM	Integer	32 bit	INIT	Global	None
			PROC1 TAB2		

DoD_SSH_050

4.3.7.3.2.2 Data Flow Analysis

The purpose of data flow analysis is to identify errors in the use of data that is accessed by multiple routines. Except for very small applications, it would be difficult to determine the data flow path for every data item. Therefore, it is essential to differentiate between those data items that will affect or control the safety-critical functions of a system from those that will not.

DFDs (Figure 4-38) should be developed for all safety-critical data items at both the module and system levels to illustrate the flow of data. The appropriate SAF should contain the product produced from DFDs. Data is generally passed between modules globally (common blocks) or locally (parameter passing). Parameter passing is easier to analyze because the program explicitly declares which routines are passing data. Data into and out of common blocks should also be traced, but additional information will have to be recorded to understand which subroutines are involved. A table should be developed to aid in the understanding of data flowing through common blocks and data passing through several layers of parameters. For each variable, this table should describe the subroutine accessing the variable and how the variable is being used or modified. The table should include all safety-critical variables and any other variables that are not clear from the DFD.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

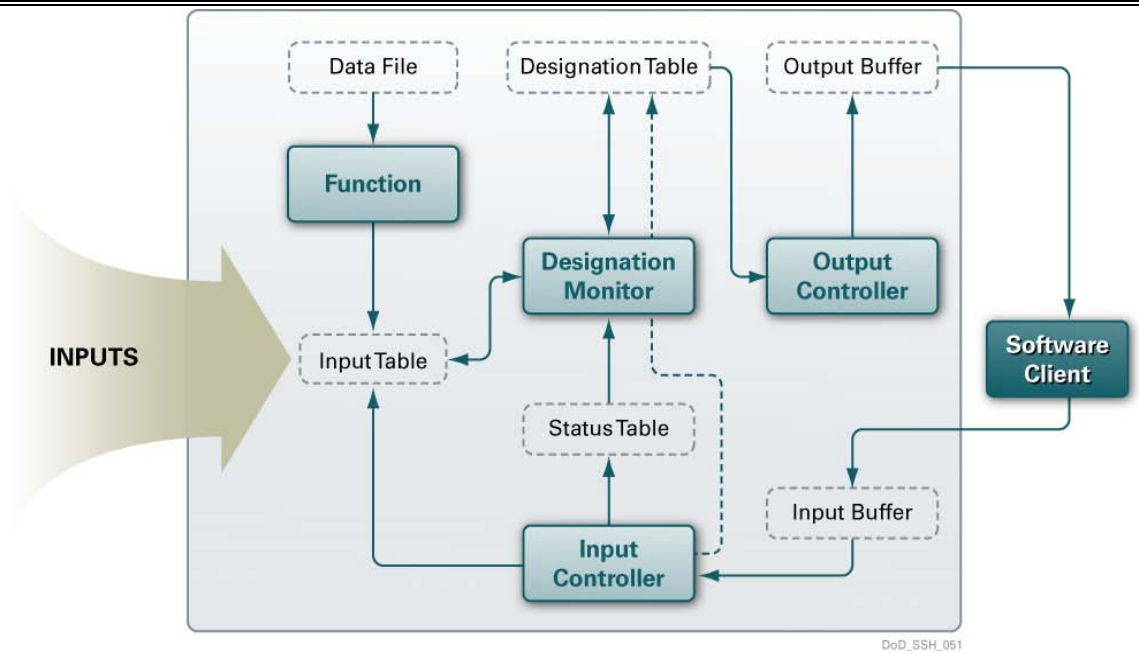

Figure 4-38: Example of a DFD

Errors that can be found from developing the data item table and the DFDs include:

- Data which is used by a system prior to being initialized
- Data which is used by a system prior to being reset
- Conditions where data is to be reset prior to use
- Unused data items
- Unintended data item modification.

It is important to analyze and test the data as it will be stored. For example, encrypting data at rest may be a classification requirement that must be analyzed and tested. Furthermore, some data calls to classified data used to mitigate a hazard may fail if the calling system is at a lower classification.

COTS can affect data in unpredictable ways if the full feature set and performance of off-the-shelf software is not analyzed and tested. For example, some COTS may manipulate data during runtime to execute a conversion (such as a global positioning system or parallax conversion), but if it is interrupted, that converted data can cause a hazard to the system if undetected and not mitigated.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.3.7.3.2.3 Control Flow Analysis

The purpose of control flow analysis (flow charting) is to reconstruct and examine the logic of the Program Design Language (PDL) and code. Constructing flow charts is one of the first analytical activities that the analyst can perform that enables the analyst to become familiar with the code and design architecture. The drawback to flow-charting is that it can be costly and time consuming. A better approach is to use the team concept and have the safety engineer's interface directly with the software developers and system engineers to understand system processing. In most cases, the software developers and system engineers have already developed control flow charts which can be made available for review as needed. Each case needs to be evaluated to determine which process would be most beneficial and cost effective to the program. In either case, flow charting should be done at a conceptual level. Each block of the flow chart should describe a single activity (either a single line of code or several lines of code) in a verbal manner as opposed to repeating each line of code verbatim. Examples of both correct and incorrect flow charts are illustrated in Figure 4-39.

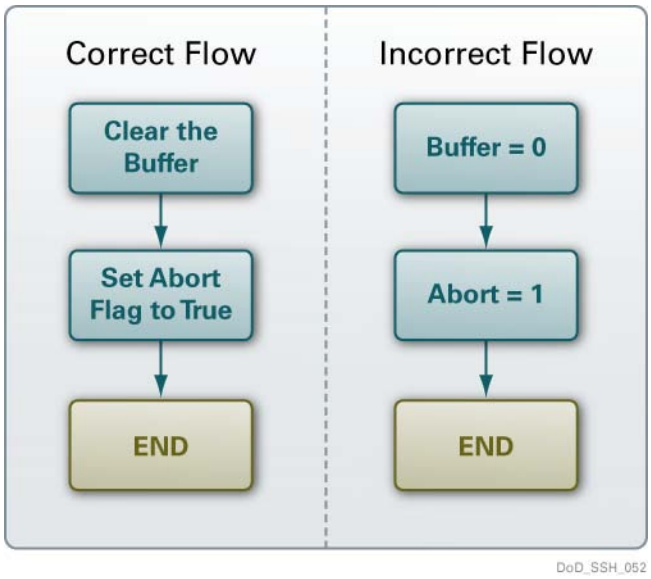

Figure 4-39: Flow Chart Examples

4.3.7.3.2.4 Interface Analysis

The purpose of interface analysis is to verify that the system-level interfaces have been encoded in accordance with IDS and Interface Control Document specifications. Interface analysis should verify that safety-critical data transferred between system-level interfaces is handled properly. Analyses should be performed to verify how system functionality will perform if the interface is lost (e.g., casualty mode processing). Analyses should also address sequencing, timing, dynamic Input/Output (I/O), and interrupt analysis issues in regards to interfaces with safety-critical functions and system components. Performing system-level testing and analyzing

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

the data traffic across safety-critical interfaces is usually the best way to verify a particular interface. Data should be extracted both when the system is under heavy stress and in low stress conditions to ensure that the message integrity is maintained in accordance with the IDS or Interface Control Document. Physical and functional interface issues in SoS environments must be analyzed for their potential contributions to failure of one or more of the SoS individual systems.

4.3.7.3.2.5 Interrupt Analysis

The purpose of interrupt analysis is two-fold. The system safety engineer must determine the impact of the interrupts on the code and the impact of prioritization of the program tasks. Flow charts and PDLs are often used to determine what will happen if an interrupt occurs inside a specific code segment. If interrupts are locked out of a particular segment, the safety engineer must investigate how deep the software architecture will allow the interrupts to be stacked so that none will be lost. If interrupts are not locked out, the safety engineer must determine if data can be corrupted by a low-priority task or process interrupting a high-priority task or process which changes the value of the same data item.

Performing interrupt analysis in a multi-task environment is more difficult because it is possible for any task to be interrupted at any point of execution. It is impossible to analyze the effect of an interrupt on every instruction. In this case, it is necessary to determine segments of code that are tightly linked, such as the setting of several related variables. Interrupt analysis should be limited to those segments in a multi-task environment. Examples of items to be considered in interrupt analysis include, but are not limited to:

- Program segments in which interrupts are locked out
- Identify the period of time interrupts are locked out
- Identify the impacts of interrupts being locked out, such as lost messages and lost interrupts
- Identify possible infinite loops, including loops caused by hardware problems
- Re-entrant code
- Are sufficient data saved for each activation?
- Is the correct amount of data and system state restored?
- Are units that should be re-entrant implemented as re-entrant?
- Code segments which are interruptible
- Can the interrupted code be continued correctly?
- Will interrupt delay time-critical actions (e.g., missile abort signal)?
- Are there any sequences of instructions which should not be interrupted under any circumstance?
- Overall program prioritization
- Are functions such as real-time tasks properly prioritized so that any time-critical events will always be assured of execution?

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Is the operator interfaces a proper priority to ensure that the operator is monitoring?
- Undefined interrupts
- Are undefined interrupts ignored?
- Is any error processing needed?
- Lessons learned.

4.3.7.3.2.6 Analysis by Inspection

Although inspection is the most commonly used method of code-level analysis, it is also the least rigorous. This does not lessen its value to the overall safety assessment process. Analysis by inspection is particularly useful for software that is less critical where a less rigorous methodology is appropriate. Analysis by inspection is also frequently used in conjunction with other techniques, such as FTAs and control flow analyses, to provide a more thorough assessment of the software. Experience shows that these analysis types are complementary. As noted earlier, a single analysis technique is rarely sufficient to meet the defined safety objectives.

Analysis by inspection is a process whereby the analyst reviews the software source code (high-level language, assembly, etc.) to determine if there are any errors or structures that could present a potential problem in the execution or the presence of adverse occurrences (e.g., inadvertent instruction jumps).

Open-source software is computer software that is available in source code form and provided under a software license that permits users to study, change, and enhance the software. Any software source code that implements safety-significant functionality must be thoroughly analyzed to ensure that it does not contribute to hazards or mishaps. The ability of the analyst to understand the code as written provides an indication of the ability of future software maintainers to understand it for future modifications, upgrades, or corrections. Code should be well structured and programmed in a top-down approach. Code that is incomprehensible to the trained analyst will likely be incomprehensible to future software maintainers. The code should not be patched. Patched or modified code provides an opportunity for a high probability of errors because it was rewritten without the benefit of an attendant safety analysis and assessment. The net result is a potentially unsafe program being introduced into a previously certified system. Patching the software introduces potential problems associated with the configuration management and control of the software.

Clue lists are lists of items that have historically caused issues (such as conditional GO-TO statements) or are likely to be problems (such as boundary conditions that are not fully controlled). Clue lists are developed over a period of time and are based on the experiences of the analyst, the software development team, and the testing organization. The list below contains several items that have been historically associated with software issues.

- Ensure that all variables are properly defined and that data types are maintained throughout the program

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Ensure that variables are properly defined and named for maintainability
- Ensure that all safety-critical data variables and processing are identified
- Ensure that all code documentation (comments) is accurate and that CSCI and CSU headers reflect the correct processing and safety-criticality of the software
- Ensure that code modifications identified by the STR have been made and the date of the modifications is noted
- Ensure that processing loops have correct starting and stopping criteria (indices or conditions)
- Ensure that array subscripts do not go out of bounds
- Ensure that variables are correct in procedure call lines (e.g., number, type, size, and order)
- For parameters passed in procedure call lines, ensure that input-only data is not altered, output data is set correctly, and arrays are handled properly
- Ensure that all mixed modes of operation are necessary and clearly documented
- Ensure that self-modifying code does not exist
- Ensure that there is no extraneous or unexecutable code
- Ensure that local variables in different units do not share the same storage locations
- Ensure that expressions are not nested beyond five levels and that procedures, modules, and subroutines are less than 25 lines of executable code
- Ensure that all logical expressions are used correctly
- Ensure that processing control is not transferred in the middle of a loop
- Ensure that equations are encoded properly in accordance with specifications
- Ensure that exceptions are processed correctly (e.g., if the “ELSE” condition is not processed, will the results be satisfactory?)
- Ensure that comparisons are made correctly
- Ensure that common blocks are declared properly for each routine they are used in
- Ensure that all variables are properly initialized before use.

The individual safety engineer can tailor or append this list based on the language and software architecture being used.

The thoroughness and effectiveness of an analysis performed by inspection is dependent on the analyst’s expertise, experience with the language, and the availability of the above mentioned clue lists. Clue lists can be developed over a period of time and passed on to other analysts. Many of the design guidelines and requirements in Appendix E and other generic software safety requirements lists are based on such clue lists. However, in Appendix E, the individual clues have been transformed into best practice design requirements.

The language used for software development and the tools available to support that development impact the ability to effectively analyze the program. Some languages, such as Ada and Pascal, force a structured methodology, strong information hiding, and variable declaration. However,

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

these languages introduce complexities and do not support specific functions that are often necessary in system development. For example, these languages allow for recursion and parallel processing in their design which may support the design requirements of the software, but may also introduce other safety issues with respect to timing and data updates. Therefore, other languages often augment them. Languages such as C and C++ support object-oriented programming and data input and output structures and provide substantial flexibility in coding. However, these languages provide for the construction of complex program statements and information hiding that are often incomprehensible even to the programmer, while not enforcing structured programming and modularization.

Some languages, such as C and C++, are more difficult to analyze than other languages, particularly those viewed by software developers as "powerful" languages. They offer significant capabilities and many ways to implement functionality, and the implementation chosen by the developer is often a personal preference. With these languages, programmers often find their own code difficult to comprehend just a few months after completing it.

Many aspects of a language affect its ability to be analyzed which include, but are not limited to:

- Language syntax
- The programming language and compilers that place constraints on the language semantics and program structure
- Data types and data constructs
- Floating point arithmetic
- Run-time bindings associated with the language.

In some languages, all of the data are available (global data) to all processes executing in the computer. Some languages allow assignment of local variables, but they are not truly local and may be available to other processes. Still other languages allow establishing local variable names that belong to the individual module and are accessible only by that module.

Array implementation is another area that often introduces difficulty in the analysis of source code. Languages have many different ways of implementing arrays and addressing locations within the array. Most languages allow indirect addresses for arrays; the results of a decision or a computation provide a pointer to a location within the array. Unless the compiler enforces memory protection and boundary protection, it is easy to overwrite arrays by writing data beyond the boundary of the array. Languages such as C-based languages do a poor job of managing these pointers and will allow the compiled program to access locations well outside the array boundaries. This can result in overwriting code or, more importantly, safety-critical data in locations outside these arrays. Unfortunately, identifying these problems at the source code level is difficult. The preferred approach is to ensure that the source code includes checks to verify that the program does not address outside the array boundary.

Memory utilization is also a critical software language factor. Memory utilization is often a function of the language that is being used, both in terms of memory management and program

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

size, and is often dependent on run-time bindings. Run-time bindings are links between modules and subroutines that occur when a program executes and may affect the analyst's ability to analyze the code. These modules and subroutines may be part of the application software, the OS or environment, or modules introduced by the compiler during compilation or COTS.

Hardware, especially the microprocessor or micro-controller, can have a significant influence on the safety of the system irrespective of the computer program. Unique aspects of the hardware may also affect the operation of the machine code in an unexpected manner. Design engineers take pride in being able to use unique characteristics of the hardware to increase the efficiency of the code or to make the reading of the machine code as obscure as possible. Occasionally, assemblers also use these unique hardware aspects to increase the efficiency and compactness of the machine code; however, it can pose limitations and possible safety risks.

4.3.7.3.2.7 Code Analysis Software Tools

Specific code analysis software tools are often used to assist in the evaluation of the correctness of the code. However, in time, specific tools mentioned in this Handbook can become obsolete or unavailable as individual companies grow, merge, or go out of business. The SSS team must evaluate the availability and effectiveness of tools prior to their use. Each tool will possess effectiveness strengths and weakness that must be evaluated and understood.

Software tools such as WINDIFF, CodeSonar, Insight by Klockwerks, and TXTPAD32 are used to aid in the code analysis process. WINDIFF, which is a Microsoft product, provides color-coded, in-line source-code comparisons for performing delta analysis. Shareware equivalents exist for this capability. TXTPAD32 is a shareware 32-bit text editor that has several features to expedite code analysis, such as Find In Files for string searches in multiple files and directories, DOS command and macro execution, bracket matching (parenthesis, braces, and brackets), line numbering, and color-coding.

The tools incorporated into the software engineering tool suite will also affect the analyzability of the source code, primarily through the information (or lack of information) that they provide regarding the source code. These tools must be identified and evaluated. Compilers and compiler tools can also affect the analyzability of the code. Most compilers offer tools that detect syntax errors; unused, unreferenced, or un-initialized variables; array boundary errors; coupling between modules; and a variety of other common errors. Many compilers offer more in terms of data protection, detection of run-time errors, infinite loops, and uncontrolled branches. Others embed functionality that removes many of the mundane tasks faced by programmers, such as garbage collection and other memory management functions. These automated and often uncontrolled processes possess the potential to introduce new mishap or hazard causal factors by delaying mitigation or even obstructing fault detection, isolation, or tolerance. Compiler optimizations also change the control structure of the source code, thus making object code inspection more difficult because optimizing compilers frequently generates different implementations for the same source code input.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.3.7.3.2.8 Formal Proofs of Correctness

Formal proofs of correctness are mathematical analyses of the source or object code that "proves" that the code actually implements the required functionality. To apply formal proofs of correctness, the developers must write the specifications in formal notation (e.g., Z, Eiffel, or gypsy). Neither formal notation nor formal proof can identify missing requirements or changes to design patterns which are new or network-based. Formal proofs of correctness of the source code provide assurance that it correctly and completely implements the given requirements; however, it does not provide assurance that the object code generated by the compiler and implemented on the processor is complete or correct. Verifying that the compiler generates the correct object code requires a formal proof of correctness on the compiler itself. After that, fully characterizing the execution of the software requires a validated mathematical model of the processor. The complexity of modern processors and features such as interrupts and exception handling make such assessments difficult. The complexity and non-determinism of networks (systems of systems) makes this task nearly impossible.

Proofs of correctness of the object code provide the assurance that the resultant executable code is complete and correct; however, they suffer from the same limitations with respect to the processor itself. Formal proofs of correctness are appropriate only for small but critical software or when mandated by Government regulations or standards.

4.3.8 System Hazard Analysis

The SHA is accomplished in much the same way as the SSHA. Mishaps, hazards, and hazard causal factors are identified; hazard mitigation requirements are identified and communicated to the design engineers for implementation; and the implementation of the SSRs is verified. However, there are several differences between the SSHA and SHA. The SHA is accomplished during the acquisition lifecycle where the hardware and software design architecture matures. Where the SSHA focuses on subsystem-level hazards, the SHA focuses on system-level hazards that were initially identified by the PHA. In most cases, the SHA activity will identify additional hazards and hazardous conditions because the analyst is assessing a more mature and integrated design than that which was assessed during the Analysis of Alternatives, CDD, and the PHA activities. The SHA activity puts primary emphasis on the physical and functional interfaces between other systems, integration of subsystems, performance during various operational scenarios, and human interfaces.

The SHA is the primary analysis activity for SoS efforts. The SHA can be used by the stakeholder prior to the RFP to identify SoS-level hazards and network-related hazards. As the SoS design architecture matures (draws near to CDR), the SHA will also become necessary for identifying, tracking, and analyzing interface hazard causal factors.

Figure 4-40 represents the primary sub-tasks associated with the SHA activity. Due to the rapid maturation of system design, the analysis performed at this time must be in-depth and as timely as possible for the incorporation of any SSRs to eliminate or control system-level hazards. As

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

with the PHA and the SSHA, the SHA must consider all possible causes of these hazards. This includes SoS hardware, software, human error, and software-influenced human error causes. As the program progresses from CDD to CDR, there are often changes in expectations as to how the system should perform in various operational scenarios, which can alter SoS-specific integration behaviors. As these expectations or behaviors change, the system specifications can be amended by adding or removing requirements. Updates to the SHA are essential to dynamically keep pace with these changes and to document SoS-related hazards involving race conditions, resource sharing, and potential deadlocks.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

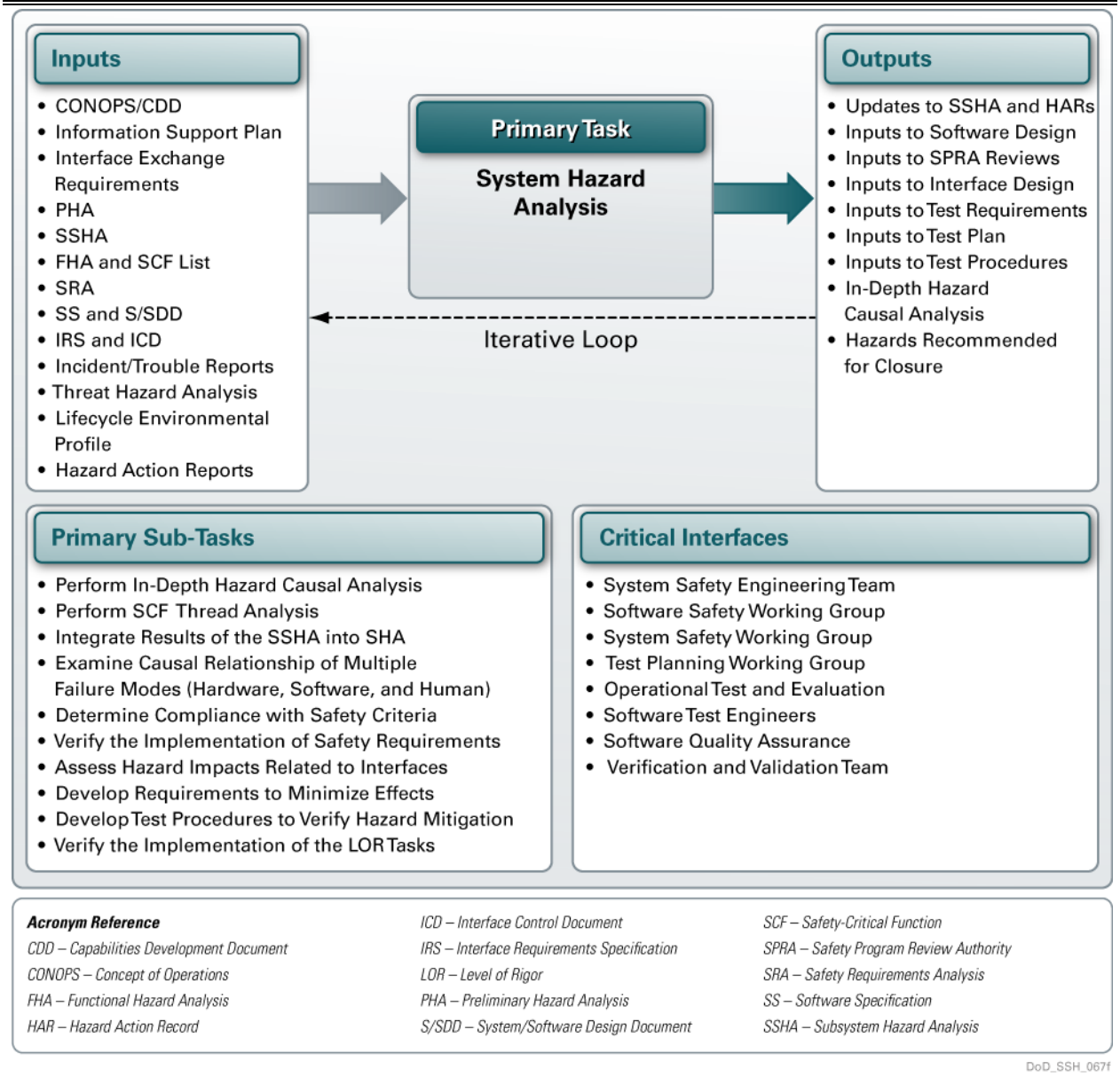

Figure 4-40: SHA

Analyzing hazard causal factors to the level or depth necessary to derive mitigation requirements will aid in the identification of physical, functional, and zonal interfaces. For the majority of hazards, the in-depth causal factor analysis will identify failure modes or causal factor pathways which cross physical subsystem interfaces, functional subsystem interfaces, and even contractor/subcontractor interfaces. The ability to identify these interfaces is depicted in the fault tree in Figure 4-41.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

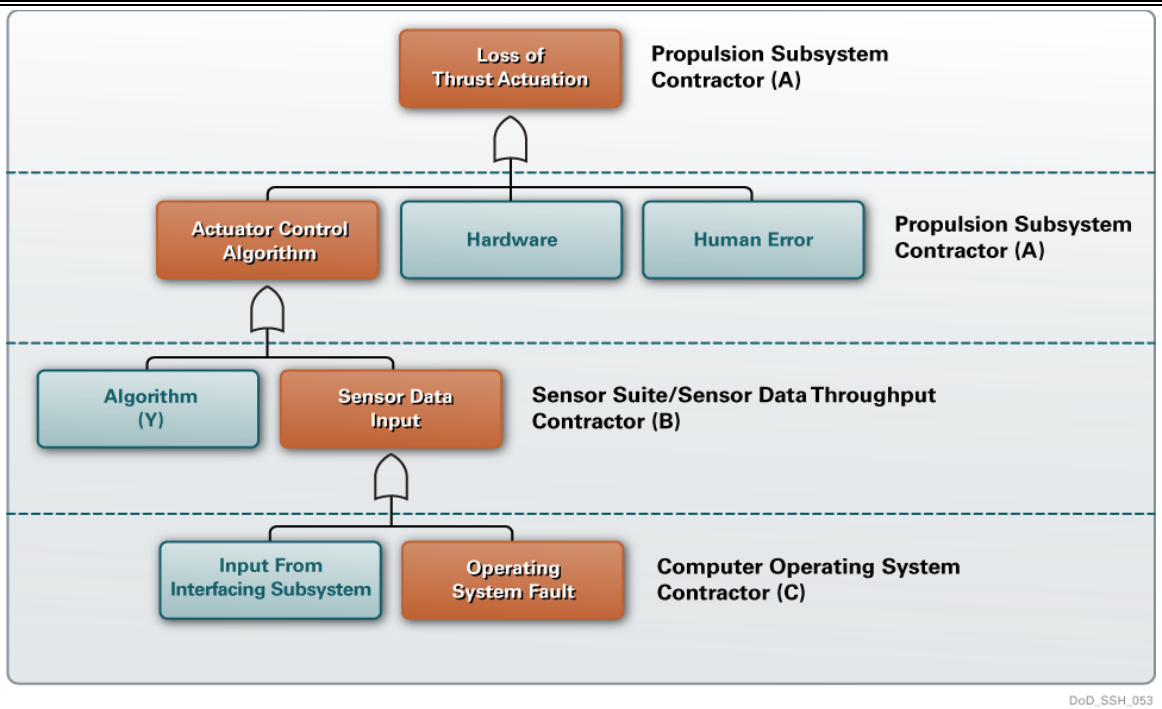

Figure 4-41: Example of a SHA Interface Analysis

In this example, the analyst uses a fault tree approach to analyze a system-level hazard loss of thrust actuation. This hazard is depicted as the top event of the fault tree. The SHA activity analyzes all potential causes of the hazard, including the software branch of the "OR" gate to the top-level event. Although this hazard may have hardware causes (actuator control arm failure) and human error causes (pilot commands shutdown of control unit), the software contribution to the hazard will be the branch discussed.

In this example, thrust actuation is a function of the propulsion system and is administratively controlled by the Propulsion IPT of Contractor A. The computer hardware and software controlling the thrust actuators are within the engineering boundaries of the same IPT. The software safety analyst has determined that a fault condition in the computer operating system is the primary causal factor of this failure mode. This OS fault did not allow the actuator sensor data to be read into sensor limit tables and then allowed an overwrite to occur in the table. The actuator control algorithm was using this sensor data. In turn, the actuator control CSC functional architecture could not compensate for the loss of credible sensor data which transitioned the system to the hazardous condition. In this example, the actuator and controlling software are designed by Contractor A, the sensor suite and throughput data bus are designed by Contractor B, and the computer operating system is developed by Contractor C.

In this example, in-depth safety analysis must be performed by Contractor C for this failure pathway to be identified. If Contractor C is contractually obligated to perform a safety analysis (specifically, a software safety analysis) on the computer operating system, the ability to bridge

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

(bottom-up analysis) from an operating system software fault to a hazardous event in the propulsion system is difficult. The analysis may identify the potential fault condition, but may not identify the system-level effects. The analysis methodology must rely on the clients of the software operating system, or Contractor A, to perform the top-down analysis for the determination of causal factors at the lowest level of granularity and then perform a communication "hand-off" to contractors B and C for further causal analysis in their domain. In the same manner but at a higher level, the SoS interfaces and IERs must be traced out to validate that the SoS hazards have been mitigated.

This paragraph will focus primarily on the maturation of the hazard analysis and the evidence audit trail to prove the successful mitigation of system, subsystem, and interface hazards. In-depth causal factor analysis during SHA activities will provide a springboard to the functional interface analysis required at this phase of the acquisition lifecycle. In addition, the physical and zonal (if appropriate) interfaces must be addressed. Within the software safety activities, this analysis primarily focuses on the computer hardware, data buses, memory, and data throughput. The safety analyst must ensure that the hardware and software design architecture is in compliance with the criteria set by the design specification. The preceding paragraphs pertaining to the PHA and SSHA (preliminary and detailed code analysis) addressed analysis techniques.

Hazard causal factor analysis and the derivation of safety-specific hazard mitigation requirements have been discussed previously in terms of the PHA and SSHA development. In addition, these paragraphs demonstrated a method for documenting all analysis activity in a hazard tracking database to provide the evidence of hazard identification, mitigation, and residual risk. Figure 4-26 (of the PHA) depicted the documentation of hazard causes in terms of hardware, software, human error, and software-influenced human error. As the system design architecture matures, each safety requirement that helps eliminate or control the hazard must be formally documented (Figure 4-42) and communicated to the design engineers. In addition, SHA activities must also formally document the results of the interface hazard analysis.

If thorough identification of GSSRs and top-level MSSRs was accomplished early in the program and these requirements were included in the SRS, the safety analyst may discover that a great deal of hazard mitigation is already present in the maturing system design. This is the result of the designer possessing a better understanding of how the system could fail and how the system design could minimize the likelihood of failure by the implementation of the top-level safety requirements that exist in the SRS. This is one of the primary reasons why the identification and documentation of GSSRs and top-level MSSRs are important in the early phases of the design process.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

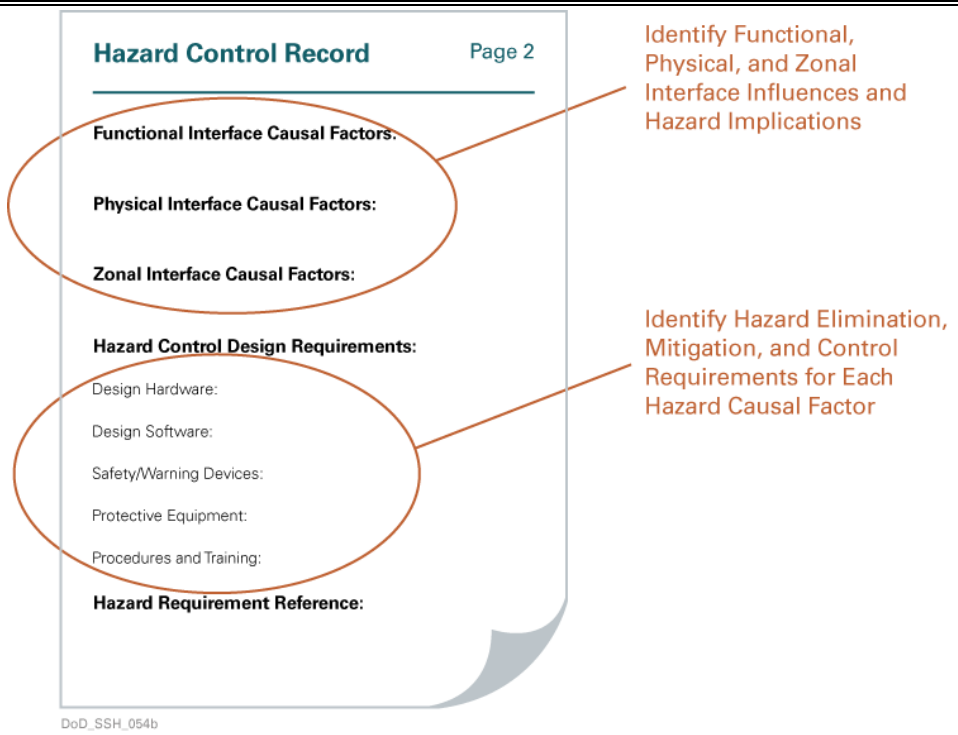

Figure 4-42: Documentation of Interface Causes and Safety Requirements

The derivation of hazard mitigation requirements from the SHA as the design matures has been a challenge to designers since the genesis of safety engineering. It is common for the designer to "design in" hazard causes late in the design process. Compounding the problem is that hazard mitigation (software) requirements must be included in the specification or they are not likely to be tested by the software test team. There is often a push-back by the software development team to add new requirements to the SRS late in the development activity. While it may be necessary to add new requirements to the system and software specifications, this activity can be minimized by writing a defect statement against an existing top-level safety requirement instead of defining a new safety mitigation requirement. This generally achieves the same outcome as writing a new requirement and is more palatable to the design team than adding new requirements late in the design process.

At this point, the safety analyst must focus on the ability to define safety test and verification requirements. The primary purpose of this activity is to provide evidence that all safety requirements identified for hazard elimination or control have been successfully implemented in the system design. It is possible that the analyst will discover that some requirements have been implemented in total, others partially, and some were not implemented at all. Active involvement in the design, code, and test activities is paramount to the success of the safety effort.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The ability to assess the system design compliance to specific safety criteria is predicated on the ability to verify SSRs through test activities. Figure 4-42 depicts the information required in the hazard control records of the database to provide the evidence trail required for risk assessment activities. Specific requirements for testing safety-significant software are identified in the LOR.

4.4 Software Safety Testing and Risk Assessment

To this point, the Handbook has focused on the analytical aspects of the software safety program. Analysis represents a significant portion of the software safety process. However, another significant effort consists of verification and validation of SSR implementation, collection of V&V evidence, and the determination and reporting of the residual safety risk.

Testing provides the evidence necessary to demonstrate that the SSRs (CSSRs, GSSRs, and MSSRs) are successfully implemented and provide the desired mishap and hazard control to include safety risk reduction. However, to provide sufficient argument that the software does not pose an unacceptable or undesirable risk, the test program must include sufficient safety-specific testing to verify that the design and implementation of the software mitigates all of the identified software-related hazard causal factors. Section 4.4.1 includes recommendations for generic tests for safety-significant software. Generic tests are structural tests relating to the selected hardware, compiler, software language, and architecture(s). However, generic tests alone cannot provide the necessary evidence that the software can safely execute in the system context. This is due to the subtle and complex interactions between the system-of-systems, control entities, the maintainer, and the software and hardware ability to control safety-significant functions.

Analysts will identify software-significant hazard causal factors during the analysis phases and develop safety-design requirements to mitigate either the hazard causal factor or the effects of the hazard causal factor. Normally, requirements-based testing will cover software safety requirements and include test cases to verify the implementation of these requirements. The safety team must review both the test plan and test procedures to ensure that the test cases provide the required evidence of risk mitigation validation. This testing should include fault insertion and failure mode testing that verifies the correct response of the software to these anomalies. As the criticality of the software increases, the depth of analysis (e.g., detailed design and code-level analysis) and the level of necessary testing increase. Analysts should use the analysis of the implementation to develop detailed test cases of safety-critical software to ensure that it achieves desired objectives.

Verification ensures that the final product (executable software application) complies with the specified requirements, and validation ensures that the software requirements correctly specify the customer needs. Verification is comprised of analysis, inspection, and testing. Hazard, requirements, design, code, and test analyses are all verification analyses which provide evidence that the correct requirements have been specified and that the software satisfies the stated requirements. In addition, requirements, design, and code inspections also provide verification evidence.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.4.1 Software Safety Test Planning

Software testing is an integral part of any software development effort. Testing should address not only performance-related requirements, but the SSRs as well. The SSS team must interface directly with software developers and the V&V team to ensure that the code correctly implements all SSRs and that the system functions in accordance with the design specifications.

Figure 4-43 provides the process activities to accomplish software safety test planning. Using best practices and contractual guidance, the SSS team should establish the required software verification activities in the verification section of the LOR table early in the program lifecycle. Table 4-4 provides an example of the test and V&V tasks that may be required for each LOR. It is necessary to establish these activities and tasks early in the program lifecycle (contract SOW and program management plans) so that sufficient resources are allocated for these activities.

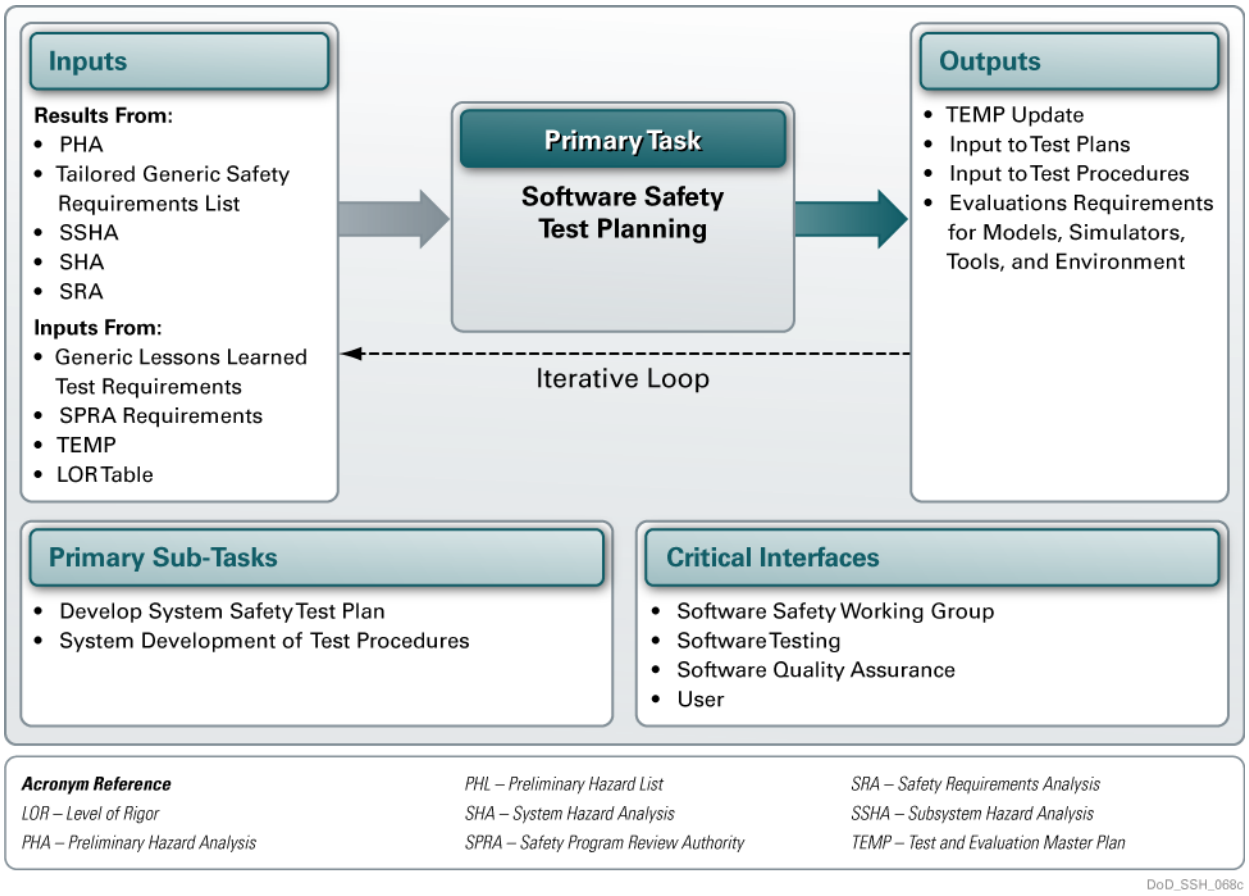

Figure 4-43: Software Safety Test Planning

At the beginning of a program, it is important to reach an agreement between the certifying safety organization, the program management office, and the system developer as to the depth of software V&V activities that must be performed to provide the desired level of confidence that

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

the software will function as expected. This agreement is documented as an integrated approach in the SEP, SSPP, SDP, SEMP, Test and Evaluation Master Plan (TEMP), and STP. The SSS team should integrate safety testing into the normal system testing effort to reduce time and cost to the program. In particular, regression testing shall be defined for the program. The extent of regression coverage of existing SSRs within any new baseline, increment, or since last regression shall be incorporated into milestone decisions. The software safety engineers, in cooperation with the V&V team, will identify the SSRs that can be verified through testing.

4.4.1.1 Guidance for Safety-Critical Software Testing

Testing safety-critical software demonstrates that the software complies with the stated requirements and shows that errors which could result in a hazard, as shown in the hazard analysis, do not exist in the software (Figure 4-44). To meet these objectives, the following guidelines should be used in developing software test cases:

- Test cases should demonstrate compliance with all software requirements (high-level and low-level software requirements)
- Test cases should verify correct functionality under normal conditions and off nominal conditions
- Test cases should ensure that all software requirements are tested (accomplished by software requirements coverage analysis)
- Test cases should ensure that all the software structures are exercised (accomplished by structural coverage analysis).

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

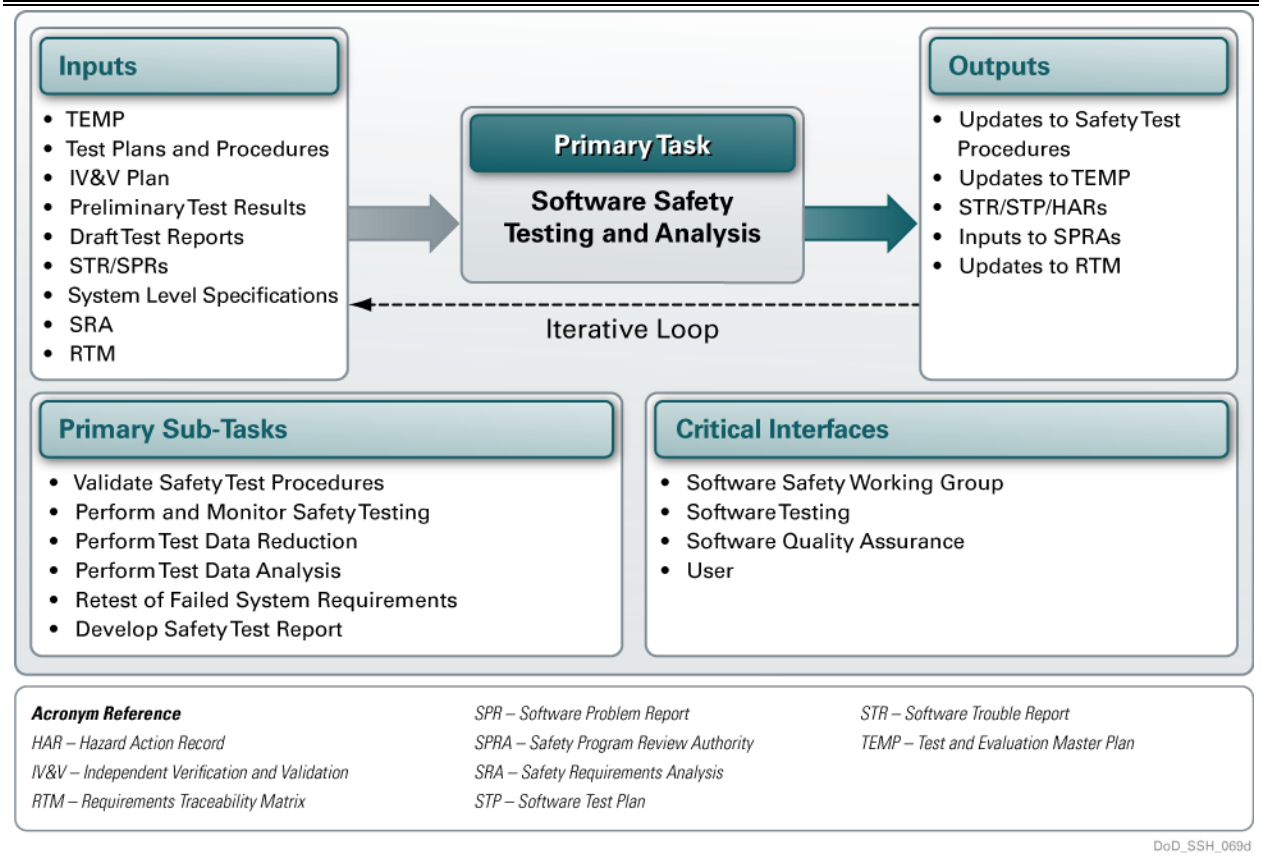

Figure 4-44: Software Safety Testing and Analysis

In terms of "off nominal conditions," systems and test engineers usually defer to "go path" requirements and typically test only the correct data that was supplied at the right time. System safety testing will exercise that right data at the right time at the system level also checks the right data at the wrong time scenarios. This same procedure should be used for data testing if the timing changes. In the context of a SoS, adding the correct user with data and time must also be tested for safe operations.

Specific tests are not mandated at each level of testing (e.g., unit testing, software integration testing, and hardware/software integration testing). However, there are instances when it is necessary to test at a lower level of integration so that test inputs, test conditions, and software behavior can be controlled or monitored to adequately demonstrate compliance with requirements; establish a level of confidence that the software is robust to error conditions; and demonstrate test coverage. Test coverage analysis must be performed to demonstrate how well the testing verified the implementation of the software requirements and to demonstrate that the entire software structure was exercised during testing.

The system safety engineer must interact with Test Planning to ensure that the proper level of confidence can be reported at significant technical reviews and milestones. Mindful of costs,

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

software model problems, and scalability, one must assure safety risk to an acceptable level of confidence by making sure that computer and network hardware, software, or firmware change(s) have not modified safety functionality due to unknown and often unforeseeable changes in sequencing, timing, saturation, robustness, and redundancies. Testing "only the changed software" is not a valid systems engineering assurance process.

The complexity of cause and effect of these combinations means that each situation is unique. Regression suite results and regression configuration comparisons are key attributes to maintaining a consistent measure of risk. The SEP, SDP, and SSPP must all define how risk will be mitigated in change assessments and V&V. Regression testing must be definitively stated as to triggers, extent, and scope and must include all safety requirements regardless of where the change occurred. Limited exceptions are permitted when trusted firewalls and validated partitions separate systems and decisions.

Safety-significant software test cases must comply with the defined requirements of the LOR tasks tailored and defined in the Table 4-4 template.

4.4.1.2 Nominal and Functional Requirements-Based Testing

Tests should be performed to demonstrate the ability of the software to respond to normal input conditions by verifying that the software complies with all software requirements. Tests should include the following test criteria:

- Exercising input variables using valid ranges of inputs and boundary values at or near the limits of the valid range of their values
- Exercising various iterations of the code to verify timing constraints are met in time-related functions
- Exercising all possible state transitions for normal operation
- Exercising logic expressions and equations to verify correct variable usage and Boolean operators
- Verifying the integrity of the partitioning of safety-critical components from non-safety-critical components
- Performing nominal and functional tests to demonstrate the ability of the software to respond to normal input conditions by verifying the software complies with high-level requirements.

4.4.1.3 Robustness Testing

Tests should be performed to demonstrate the ability of the software to respond to abnormal inputs and conditions. In this context, MIL-STD-1679 is still a valid reference resource. Tests should include the following criteria:

- Exercising input variables using invalid values and out of range boundary conditions

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Performing system initialization under abnormal and stressing conditions
- Exercising the software using possible failure modes of incoming data as determined by the FHA or software failure modes, effects, and criticality analysis
- Exercising software loops to out-of-range loop count values to demonstrate the robustness of loop-related code
- Exercising software to ensure protection mechanisms for exceeded frame times respond correctly and as predicted
- Performing tests to exercise arithmetic overflow protection mechanisms
- Exercising the software to invoke state or mode transitions that are not allowed by the software requirements
- Performing stress testing where inputs are varied to exceed the limits specified in the SRS and interface requirements specifications and force system anomaly conditions (e.g., division by zero); stress testing is performed to verify limits of safety-critical modules (e.g., CPU usage, input/output response time, memory use, and network utilization)
- Performing robustness testing to demonstrate the ability of the software to respond to abnormal inputs and conditions by verifying the software complies with the high-level requirements.

To demonstrate adequate test coverage, test coverage analyses must be performed. Test coverage analysis techniques, such as Modified Condition and Decision Coverage (MC/DC), are commonly misinterpreted as actual tests rather than analyses. These analyses determine how well the testing verified the implementation of the software requirements and demonstrate that the entire software structure was exercised during testing. Requirements coverage analysis and structural coverage analysis must be performed for the appropriate LOR, as specified by the program-established LOR (see Figure 4-13).

4.4.1.4 Requirements Coverage Analysis

Requirements coverage analysis should be performed as specified by the program-established LOR. The requirements coverage analysis should demonstrate that:

- Test cases exist for each software requirement
- A test case exists for each high-level software requirement and that test cases satisfies the criteria for nominal and robustness testing
- Test cases satisfy the criteria of nominal and functional requirements-based testing and robustness testing
- An analysis was conducted to verify test coverage for low-level software requirements
- A test case exists for each low-level software requirement and that test cases satisfy the criteria for nominal and robustness testing.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.4.1.5 Structural Coverage Analysis

Structural coverage analysis should be performed as specified by the program-established LOR (see Table 4-3). The different levels of structural coverage are:

- MC/DC demonstrates that every condition for a decision in the program has exercised all possible outcomes at least once, and every condition in a decision has been shown to independently affect that decision's outcome
- Decision coverage demonstrates that every point of entry and exit in the program has been invoked at least once, and every decision in the program has shown all possible outcomes at least once
- Statement coverage demonstrates that every statement in the program has been invoked at least once
- Semantics of the programming language are such that the proper use of the language and its data structures and tables are correct.

Structural coverage analysis is also used to confirm data coupling and control coupling between the code components.

4.4.1.6 Formal Safety Qualification Testing

Formal software qualification consists of a specific suite of test cases that are used to provide evidence to the SSS team for safety certification. Formal qualification testing should be performed at the highest level of integration for each CSCI because tests run on the entire software application (all modules compiled together and all software components interacting with each other in the tactical hardware) provide the highest level of fidelity for the actual hardware/software configuration that will be running in the field. However, to achieve the necessary coverage (i.e., being able to exercise all the code), it is sometimes necessary to use lower-level test cases (even down to unit testing) for formal qualification. Regardless, test cases should be documented, including the level of integration required to achieve the necessary level of coverage. These test cases should be noted as safety critical.

It is also important to establish a safety regression philosophy. As the code is baselined and undergoes qualification testing, the software may need to be reworked as a result of issues discovered during successive levels of integration. Regression test criteria include:

- Perform regression testing for each new build to verify that subsequent builds do not impact the previously tested and qualified safety-critical functionality
- Perform safety regression testing for software modifications or revisions within a build that has been modified after formal qualification
- Create a minimum set of unit, unit integration, and CSCI-level software qualification regression test cases for testing all safety-critical software functionality.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The software test planning process (refer to Figure 4-43) must address all simulators, models, emulators, and software tools that will be used by the test team (whether for V&V or safety testing) to ensure that all processes, requirements, and procedures for validation are in place. This validation must occur prior to use to ensure that the data is processed as intended. Invalid tools will invalidate the test results. The software safety test plan addresses how the software safety engineer will participate in Test Working Group (TWG) meetings and how inputs will be provided to the TRR and SRB.

Outputs from the software safety test planning process include updated V&V plans; updates to the Test and Evaluation Master Plan; and evaluation requirements for simulators, models, emulators, test environments, and tools. Outputs also include updates to the Software Test Plan and recommendations for modifications to test procedures to ensure coverage of all SSRs identified for test by the RTM and software verification trees.

The safety manager must integrate software safety test planning activities into the overall software testing plan and the TEMP. This integration should include the identification of all safety-critical code and SSRs. The SRA must include all SSRs. The software safety test planning must also address the testing schedule (functional qualification testing and system-level testing) for all SSRs. This schedule must be integrated into the overall software test schedule and the TEMP. The software safety test schedule will depend on the software test schedule and the safety analysis. Beware of test schedule compression due to late software development; the safety schedule must allow sufficient time for the effective analysis of test results. The SSS team must identify system-level test procedures or benchmark tests to verify that the hazards identified during the analyses (SHA and SSHA) have been eliminated, mitigated, or controlled.

During the software safety test planning process, system safety engineer/analyst/team/etc must update the RTM developed during the SRA by linking requirements to test procedures. Each SSR should have at least one V&V test procedure. Many SSRs will link to multiple test procedures. Linking SSRs to test procedures allows the safety engineer to verify that all safety-significant software will be tested. If requirements exist that system safety engineer/analyst/team/etc cannot link to existing test procedures, the safety engineer can either recommend that the SSS team develop test procedures or recommend that an additional test procedure be added to the V&V test procedures. There may be cases where SSRs cannot be tested due to the nature of the requirement or limitations in the test setup. In this case, software-detailed design analysis must be performed.

The SSS team must review all safety-significant V&V test procedures to ensure that the intent of the SSR will be satisfied by the test procedures. This is also required for the development of new test procedures. If a specific test procedure fails to address the intent of a requirement, the SSS team is responsible for recommending appropriate modifications to the V&V team during TWG meetings.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.4.2 Software Systems Safety Tests

Software testing generally occurs in phases from unit-level testing through integration testing, system integration testing, acceptance testing, and operational evaluation. Depending on the nature of the software and software development process, unit-level and integration testing may overlap. Some units may undergo initial testing when others are undergoing integration testing. Testing processes, such as regression testing, will occur during most phases of software testing. .

Software systems safety testing includes a variety of test types, many of which are common testing techniques, others which are uncommon for many test programs. The ultimate concern is the effect of software and software errors or failures at the system level. Software itself is not hazardous until interfaced with a hardware system with associated hazards. Therefore, the focus of much of the software systems safety testing is on the system-level effects. Safety testing is iterative in nature during all phases and is an integral part of all test processes. Testing at the unit level may result in the addition or modification of tests at the module level. Conversely, anomalies discovered at the module level will likely result in changes to software units requiring retest. These iterations occur at every level of testing through acceptance testing and operational evaluation. However, by the time the system reaches the operational evaluation level, all safety-significant anomalies should be corrected or the system will receive an unfavorable evaluation.

Common testing techniques include requirements-based tests, functional tests, path coverage tests, statement coverage tests, stress tests, endurance tests, and operator interface tests. Some of the less common test types include fault insertion and failure mode (hardware and software) tests, GO/NO-GO path tests, boundary condition tests, and operator error tests. Other test techniques include perturbation and mutation testing. A requirement for testing all safety-significant software is that there be a formal test coverage analysis to verify that as many conditions as possible are subject to test in the safety-significant software.

The RTM is an important tool in the software systems safety testing process. This matrix documents full test coverage of safety-significant software whether a safety-significant function is an inherent part of the system design or is an SSR designed to mitigate an identified hazard causal factor. Full test coverage will likely require a combination of the previously mentioned testing techniques.

Software safety test programs should be designed such that errors are found at the appropriate level. Coding errors and errors in individual units should be found during unit-level code reviews and testing. Finding these errors in later test evolutions increases the cost of correction and may result in schedule impacts. Therefore, the safety team needs to design the test program to find errors as early as practical in the test evolution.

4.4.2.1 Requirements-Based Tests

Requirements-based tests focus on verifying that the software implements the high-level requirements in the software requirements specification and software design documents. These

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

tests include safety design requirements and other software safety requirements from the SRS. The test team will develop test cases and procedures that verify and validate the implementation of the requirements. However, like the development process, the wording and intent of the requirements is subject to interpretation. Therefore, the SSS team must provide guidance to the test team on the intent of the safety design requirements and review the test cases and procedures to ensure that the results will verify and validate the software safety requirements. The SSS team must also provide guidance on the testing of other software safety requirements to ensure that the test team can identify potential safety anomalies in their execution. This will require the SSS team to participate in the development of test cases and procedures and review the results from any test oracle[13] used.

4.4.2.2 Functionality Tests

Functionality tests are similar in nature to the requirements-based tests; however, the focus is on the system-level functionality vice the specific requirements. Whereas requirements-based tests may not verify that the functionality designed into the software is correct, functionality tests do not necessarily verify implementation of the SRS requirements. Therefore, these test techniques are complementary.

Application of functionality testing generally occurs later in the testing phases when the modules that form the functional thread undergo integration testing. The results verify that the software implements the system-level functions required for the system to perform its mission. Functionality tests may not adequately test software safety requirements, especially safety design requirements. Functionality-based tests are more likely to identify design requirements that are unnecessary (i.e., validate the design) to the execution of the system functions. However, if the SSS team identified the safety design requirements and designated the software safety requirements in the software documentation, the testing team will fully understand that safety design requirements are not unnecessary requirements.

4.4.2.3 Path Coverage Testing

Path coverage testing designs test cases and procedures to ensure that every possible path through the source code is exercised at least once. Contracts will occasionally specify that the software testing must include 100 percent path coverage during testing. Complete path coverage testing can be challenging because the complexity of the integration increases. Testers most often apply path coverage testing during the unit-level and integration tests. Path coverage testing often requires the use of failure mode testing, fault insertion testing, and occasionally mutation or perturbation testing. The failure mode and fault insertion testing allow the test team to exercise paths that may only occur if these conditions are present. Similarly, other paths may

13 A test oracle is a program or means used to determine what outputs should be expected from a given test. The test oracle may be a look-up table for mathematical functions, a mathematical model executed on another computer, or hand calculations that identify a solution. Validation of the test oracles used for safety-related software is an essential part of the safety testing process.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

require mutation or perturbation tests to ensure their coverage (see Sections 4.4.2.11 and 4.4.2.12).

4.4.2.4 Statement Coverage Testing

Statement coverage testing ensures that every statement in the source code is executed at least once during testing. Although similar to path coverage testing in its processes, statement coverage testing does not provide as comprehensive a test series. The test can exercise a statement without necessarily exercising all of the paths that result from it. For example, a test may exercise an IF statement where one path may bypass execution of the subsequent lines of code to the end point of the IF statement. Statement coverage testing would require only that the "success" path through the IF statement be exercised, whereas path coverage testing requires that both paths are executed. Statement coverage testing may not identify latent faults as effectively as path coverage testing.

4.4.2.5 Stress Testing

Stress testing focuses on determining the ability of the system to function under high stress conditions. Stress testing usually occurs during or just prior to system integration testing. High stress conditions for software include high throughput on data buses, input/output channels, memory, operator throughput, etc. All of these conditions can result in anomalous behavior by the software. Timing issues often occur during high stress conditions that lead to undesired behavior. Physical conditions, such as high temperatures or electromagnetic interference, may be part of stress testing to see how effectively the system can handle these conditions. Occasionally, very low stress conditions can also lead to anomalous behavior and must be part of the stress testing as well.

4.4.2.6 Endurance Testing

Endurance testing demonstrates the ability of the system to run for a defined period of time without failing or requiring a warm start. Contracts frequently require that systems undergo endurance testing for a specified time, and will often specify acceptable actions that operators may take in the event of failures. Older Military Standards for software development included endurance testing requirements for all software, especially software used for command and control, weapon control, and fire control. The general requirement was that the software execute for 24 hours without requiring a cold or warm start. The baseline for this requirement was that older computers were relatively unreliable and software tended to have many errors that would result in a crash. Most testing organizations performed minimal operations during that 24 hour period, making the test easier to pass. Modern processors are capable of extended operation. Therefore, endurance testing for 24 hours is easy to achieve as long as the software does not fail. Most contracts that require endurance testing specify longer time periods and require that the system be operated in a fashion similar to what it would experience in operational use.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

While endurance testing is not specifically safety related, the anomalies discovered during the test are as important as those discovered during other testing. The SSS team and test team must evaluate these anomalies for potential safety anomalies or conditions that may be indicative of potential risks.

4.4.2.7 User Interface Tests

User interface tests include a variety of tests designed to verify the functionality of the operator interface, the maintainer interface, the adequacy and completeness of information displays, the "intuitiveness" of the operator and maintainer controls, and human factors considerations. While much of the system integration testing requires use of the operator interface, this testing specifically focuses on the user interface(s).

Many modern system developers will begin this process before the software design is mature. The prototype user interface is often developed with test stubs to react to the operator's inputs and display dummy information. Developers will allow experienced users to use the interface as if operating the actual system. Operators will frequently make recommendations for changes to displays, including the addition or deletion of data displayed on the interface. Users will also make recommendations for changes to the controls to make the interface more user friendly, more compatible with existing systems, or to perform necessary operations that may not be in the interface specifications. The user interface prototypes are typically sophisticated software packages that may or may not serve as the baseline for the user interface in the final system.

Human Factors Engineering (HFE) is frequently a part of the user interface design process. The SSS team should establish close ties with the HFE team as part of the user interface evaluation. The SSS team can provide the HFE team with many of the safety-significant concerns and allow HFE to address these issues during the evaluation process. Part of the safety evaluation is the display of safety-significant information required by the operator. The HFE team can best determine which display formats are most beneficial and likely to provide proper recognition of the necessary safety-significant information by the operator. Unnecessary information can result in confusion or cause the user to miss important data. Therefore, the assessment of the displays must include assurance that the necessary information is available, and also that extraneous information does not interfere with the safe operation of the system.

Design requirements for safety-significant controls must be verified during the evaluation. For example, a typical generic requirement is that the operator be able to exit any safety-significant routine with the activation of a single button, key, or other control. However, if the "cancel action" control is not apparent to the operator, hazardous conditions may result. HFE can provide the best evaluation of control location and recognition.

Two subsets of user interface tests are critical to the overall safety evaluation of the system—operator error testing and free-play testing. The latter, as its name suggests, allows the test operators to function the system without scripted procedures. The users will frequently push the operator or maintainer interface to its limits to see how the system reacts. Although the tests are not scripted, data extraction tools often track the operator and maintainer actions such that, if

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

anomalies occur, the test team can repeat the sequences of operator actions to determine if the anomaly is an error in the user interface, an error in the software, or a transient error. Testers will repeat these procedures after any fixes to verify that the fix corrected the problem.

Operator error tests are intentional erroneous entries used to verify that the system responds safely and correctly to these errors. For example, the operator may enter a value when prompted that is outside the expected range of values for the system (e.g., a longitudinal position of 370°). The interface design should reject the input and alert the operator of the error. Errors may also be errors in the sequence of operations, such as trying to commit a weapon to fire before arming the system. The objective is to identify any user errors that cause the system to behave in an undesirable fashion. Similarly, testers can intentionally enter erroneous data on the system maintenance interface to determine the reaction of the system. Such erroneous entries may include invalid passwords for password-protected access; attempts to bypass security features; and errors downloading data, especially new program loads.

Several of the generic safety design requirements in Appendix E address the user interface. The safety team should identify these requirements and ensure that the test team verifies that these requirements are properly implemented. If the system does not achieve the intent of the requirement, the safety team will need to assess the potential risk and recommend acceptance of the associated risk or changes to the interface.

4.4.2.8 Fault Insertion and Failure Mode Tests

Fault insertion and failure mode tests provide assurance that the software will safely respond to various faults or failures in the hardware and software. Fault insertion testing may involve modifications to interfacing units (hardware or software) or modules to determine the reaction of another unit, module, CSCI, or the system to that fault. For example, if a software system includes a unit performing a cyclic redundancy check on safety-significant data received from an interface driver, the SSS team will want to test that unit with a test stub capable of allowing them to input erroneous CRC checksums. The SSS team will also test the interface driver to ensure that it responds correctly to notification by the CRC routine that the check failed. Both of these are examples of faults.

Failure mode testing involves creating failure conditions and determining the response of the software. Unless the development team designed the software to be robust (i.e., respond correctly and safely to failures), it is likely that these tests will result in a failure of the unit, module, or CSCI, or the failure will propagate through the software resulting in anomalous behavior of other software or hardware elements of the system.

The SSS team should identify those faults and failures that could adversely affect the safe operation of the software and develop mitigation recommendations to ensure the system responds safely to their occurrence. Fault tree analyses and similar detailed analysis techniques are excellent tools for identifying such faults; however, application to software is time consuming and may occur too late in the development cycle to be useful for developing fault

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

mitigation. Therefore, the SSS team may use less rigorous approaches and identify faults in a more generic manner.

For example, if a unit interfaces to an input sensor (e.g., an analog to digital converter monitoring a safety-significant input), the safety team may identify potential failure modes of that sensor (e.g., a large change in the values between readings) and develop a design requirement to mitigate any potential risks (e.g., if $r_x > (r_{x-1}+0.1r_x)$, then reject the reading). Other failures are more obvious, such as a FIRE signal being present before the ARM signal is generated. This may indicate an operator error or a hard failure in a FIRE push button (whether implemented in hardware or software).

In some cases, hardware interlocks require testing to determine both their effectiveness as an interlock and the system's response to failures of that interlock. An example of such an interlock is a Navy requirement to have a hardwired key switch that enables or safes (prevents launch) weapons. A key under the control of the Commanding Officer or his designee provides a single point of control over the firing of the weapons. Although the key switch is hardwired, the software must monitor that key switch such that it does not attempt to execute an undesired action. The key switch should output two signals in each state—Weapons Tight and Weapons Not Free (in the "safe" state) and Weapons Free and Weapons Not Tight (in the enabled state). Any other combination of input signals indicates a failure in either the key switch or the software reading the key switch. The SSS team must decide the acceptable response to a failure of the key switch and provide safety design requirements or recommendations to mitigate the potential risk.

Fault insertion and failure mode testing can be intensive and time consuming. Therefore, the SSS team must focus those tests on areas that will provide the greatest overall benefit and risk reduction.

4.4.2.9 Boundary Condition Tests

Boundaries, especially the numerical value zero, pose special problems for software. Except for the number zero, most boundary conditions are artificial conditions set up in the software.[14] Where the software defines a boundary during initialization, such as a data array or protected memory, software that accesses these areas must incorporate checks to ensure it does not exceed the boundaries of the array or memory or violate the protection of the memory. When the latter two occur, the operating system or processor may throw an exception. Therefore, the system software should also include exception handlers for these violations.

Boundaries on input or output values should take the form of a reasonableness or "sanity check" on the data. If the value is outside the acceptable range, the software should execute a recovery routine designed to minimize the adverse effects of the input or output, including safety-significant effects. If the inputs or outputs consistently fall outside the acceptable range, the software must flag that interface as failed. The SSS team should identify those parameters that,

14 Memory boundaries may be physical or software controlled.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

should they fall outside the acceptable or expected range, will lead to a hazardous condition and provide that data to the test team. As early during development as practical, the safety team should provide those parameters to the software developers, along with the recommended actions should the software detect a hard failure. Although this seems to occur late in the development cycle, many of the parameters can be determined earlier in the design process once the software development team refines the architecture to the point of defining modules. Many of the input and output functions and the acceptable limits on the associated data will be known. If incorporated during the early design phases, the development team can modify the upper and lower values for the sanity or reasonableness checks with little programmatic impact. If these checks are not part of the initial design, incorporation of these checks may result in programmatic delays and cost overruns. Safety testing should include values on either side of the boundary, the value of the boundary itself, and values approaching the boundaries.

Computer programs frequently embed models to perform certain types of calculations or obtain specific data. For example, a long range missile or aircraft that flies close to the ground will have to model the earth's surface such that it can maintain a safe altitude throughout flight. There are many ways to maintain the safe altitude—model the earth's surface, use a look-up table that shows the elevation of various coordinate points on the surface, use a radar altimeter, etc. In general, aircraft will use a combination of methods. Although radar altimeters can help maintain a safe altitude, they require a backup system should they fail or encounter conditions that preclude their use. Radar altimeters also emit a signal that an enemy could use to target the aircraft or missile. Therefore, another method of maintaining altitude is required. Look-up tables and models require up-to-date information regarding the terrain and any obstructions that may protrude into the aircraft or missile flight path.

Modeling spheres in software is challenging. Modeling spheres with protrusions and changes in diameter is even more difficult. Therefore, models of the earth tend to employ adjacent plates. Each plate is a flat surface that covers a small area of the earth's surface. The angles between the plates are typically small; however, they represent a boundary that the test team must test as part of the overall test program. The plate's position and altitude ensure that it is above any obstructions that may protrude into the flight path. Note that plates are not always orthogonal to the earth radial. Mountainous terrain will require that these plates reflect the changes in the shape of the earth's surface; hence, they will have various angles to the radial. Whether the missile or aircraft is flying over mountainous terrain or flat land, there will be a finite angle between the plates. The transition between plates represents a boundary that the software must accommodate. One missile system[15] failed to adequately account for this change. The missile used a model similar to the one described in combination with a look-up table. During flight, the missile maintained a relatively consistent altitude above the surface; however, when it encountered an exception when reading from the look-up table, it reverted to the model. This occurred as the model was approaching a boundary between plates. As the missile transitioned across the boundary, it adjusted its angle of attack to maintain the desired altitude. Unfortunately, its altitude was already correct and the correction made to achieve the altitude commanded by the model resulted in a crash. This example demonstrates the need to test these

15 There are two similar reported incidents involving independently-developed missiles in two different countries.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

boundary conditions, approaching the boundary from both sides as well as the boundary value itself.[16]

Array and other memory boundaries are a common source of issues in software. In general, designers specify an array or other memory blocks with limits on the size specified during initialization. However, other routines may either overwrite the array or access data that is outside the array. Since data outside the array can be indeterminate, any calculations using that data are likely erroneous. Some compilers, such as Ada compilers, will ensure that units and modules cannot overwrite array or memory block data or access data outside the array or memory block. However, most other compilers have no such capabilities. In addition, languages (especially the C-based languages) have known issues with memory management and memory pointers. Therefore, software developers must exercise care when designing software containing arrays or defined memory blocks where units and modules under development by the various groups cannot go outside of or overwrite these boundaries. In C-based languages, this requires attention to the definition of the arrays and memory blocks and to pointer arithmetic. From a safety perspective, the SSS team can do little other than ensure that the SQA group is aware of the issue and maintains vigilance over the software development processes, especially as they relate to arrays, memory blocks, and pointer arithmetic.

A special case is the use of memory blocks to store large segments of data, such as telemetry data. The routines that gather and store this data are the most common culprits of array or memory overwrites and out-of-bounds conditions. An instance of this occurred during one of the Apollo missions to the moon. Appendix E discusses this instance.

The number zero occasionally causes issues in computers. Some processors have two representations for zero plus zero (binary all zeros) and minus zero (either binary all ones or all zeros, with the most significant bit set to one). This may cause problems when the computations transition across the threshold. This can also create issues when software designers start writing in-line code.[17] Most software developers do not understand how the host processor functions or how the processor used for development may vary from that used in the actual system. These issues can also create problems when the compiler used on the source code does not specifically target the processor in the system. Where safety-significant functions may involve a transition across the zero boundary, the SSS team should ensure that the tests include cases where the values approach zero from both sides, transition across the zero boundary (from both directions), and the value zero itself. These tests will be similar to other boundary testing.

4.4.2.10 GO/NO-GO Path Tests

Computer programs frequently have two success-oriented paths (GO paths) through the functions and corresponding NO-GO paths. The latter are those conditions that preclude the system from accomplishing its mission or cause degradation of mission capabilities. Many of

16 It is not uncommon for designers to omit the boundary value in the calculations.
17 In-line code refers to a block of assembly code or a binary executable embedded in sources code. Many high-level languages permit this practice.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

these conditions are safety significant. This is especially true if the SSS team incorporated safety-interlocks into the software design to preclude certain hazard causal factors. Testing, especially at the system integration level, is frequently success oriented (right data/right time), with the goal to prove that the system software achieves its required functionality. There is a strong focus on ensuring that the software achieves the mission objectives as accurately and efficiently as possible. This approach can lead to inadequate testing of the NO-GO paths (wrong data/right time, right data/wrong time, or wrong data/wrong time). The SSS team must ensure that the test cases include adequate coverage of the safety-significant NO-GO paths as well as the GO paths. This may involve creating conditions that cause the software to fail, subsequently requiring the use of fault insertion or failure mode testing. This testing may also include preventing inputs to the software that satisfy the conditions for the GO path.

4.4.2.11 Mutation Testing

Mutation testing modifies the code in a module or unit to achieve a specific testing objective. As a general rule, modification of the code under test invalidates the test; however, mutation testing is necessary in certain cases. For example, OOA&D does not permit visibility into specific instances of objects because the instantiation of the object[18] changes some of its characteristics and attributes. Therefore, testing the functionality requires a "black-box approach,"[19] an undesirable testing process. From a functionality perspective, this may be acceptable; however, from a safety perspective it is not.

Mutation testing introduces minor modifications to the object such that when it is instantiated and executed, test personnel can observe certain features, behaviors, or results of the execution. For example, test personnel can introduce a test stub that causes the module to output an intermediate value during the execution of the module. By comparing that value to the results of an oracle-generated value, the test team can determine whether calculations to that point are correct. Another application of mutation testing is verifying that an object behaves as desired when certain parameters, such as adaptation data change. If the routines providing adaptation data are not available and tested, mutation of the object by introducing the adaptation data is an acceptable means of executing the test and verifying the response of the object (and its interfacing objects, when available) to changes in the adaptation data.

Mutation testing can also involve the modification of one software unit to determine the effects on another software unit. For example, a routine receives data from another routine that it uses to perform mission-critical or safety-critical processing. Due to the critical nature of the process,

---

18 Instantiation of an object refers to loading and enabling of an instance of a function (object). Memory parameters, I/O addresses, calling routines, and called routines are defined when the object is instantiated. Other parameters, such as system adaptation data or specific functional capabilities, may also be established at the time of instantiation.

19 Black box testing refers to the process of providing inputs to a unit under test (hardware or software), observing the outputs, and determining the internal functionality from the relationship of outputs to inputs. For linear systems, this is a valid approach to verifying functionality; however, for discrete digital systems, it is not. Discrete systems can make infinite changes in the output state with small changes in the input state.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

the routine implements a checksum to verify the validity of the incoming data. However, unless a bad data stream arrives across the interface, verifying the correct implementation of the checksum algorithm may be difficult. Two approaches can be used—cause the interfacing unit to generate a bad data stream (i.e., mutate the output of the other unit, a form of failure mode testing) or develop a stimulator that allows the generation of a bad data stream. Developing a stimulator to input a bad data stream is only a little more complex than generating a stimulator to generate a good data stream. However, the ultimate non-operational test is interfacing the software unit with the one that it will interface within the system in an environment that simulates the operational environment as closely as practical. Mutation testing allows verification that the unit under test will react as desired to the invalid data input. There are other variations of mutation testing that can provide valuable insight into the execution of software developed using OOA&D, component-oriented design, and package-oriented design.

4.4.2.12 Perturbation Testing

Perturbation testing is a variation of mutation testing in which the test team "perturbs" the execution environment to determine the reaction of software. Testers first applied perturbation testing to identify security flaws in a software package executing on an operating system. By introducing perturbations into the OS, they were able to take advantage of access paths in the application software that were not available under normal conditions. Later applications of perturbation testing found a significant safety flaw in the automatic braking system software used in many automated subway systems.

Perturbations in the operating environment can take many forms, including forcing exceptions errors in I/O handling, corruption of communications between software modules, and forcing hard and soft interrupts. By forcing exceptions, testers can verify that the application software recognizes and properly handles the exception. Similarly, testers can force the occurrence of certain interrupts to determine the response of the software to their occurrence. This is particularly important during system integration testing where the ability to force these errors via software is lost.

4.4.2.13 Test Phases

An important issue to consider during the testing phases is ensuring that the tests are designed to find anomalies at the appropriate level. Errors in logic, errors in coding, and requirements implementation errors should be revealed during unit-level code reviews and testing. Locating such errors in higher-level tests may result in programmatic delays because the affected unit must undergo modification and the previous levels of testing again before the testing can resume. Therefore, the safety team must design tests to ensure that they identify and isolate safety-significant errors at the lowest level test series practical.

A unit is the smallest distinguishable software component that performs a function, such as an interface driver. Modules generally consist of two or more units and perform a higher-level function, such as signal processing. CSCIs consist of one or more modules and generally perform a segment of system functionality, such as engagement processing. The types of tests

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

performed vary with each testing phase, although testers can apply almost any of the techniques at any phase.

Unit testing is the best opportunity to perform failure mode and fault insertion testing where the functionality contained in the unit must respond correctly and safely. An example is a cyclic redundancy check processing routine where entering erroneous data at the unit-level is easier than at higher levels of integration. However, the interface drivers (simulators) to the unit must provide an accurate model of the data transfer functionality of the unit that interfaces to the CRC processing routine. During integration testing, forcing erroneous data into the CRC processing routine requires forcing an error in the associated interface driver (see the discussion of mutation testing), a much more difficult task.

Integration testing begins with interfacing units to create modules. If the units underwent sufficient testing and any used test stubs were validated, the integration testing should proceed smoothly. However, integration testing is where function-related errors, timing issues, and latent requirements implementation errors begin to appear. Therefore, much of the testing focus, from both safety and software perspectives, is in identifying and correcting these deficiencies. During integration testing, functional and requirements-based testing begins in earnest. This is the best opportunity to execute GO/ NO-GO path tests, boundary condition tests, and fault insertion and failure mode tests. In some cases, mutation testing may be required during integration testing to provide the evidence necessary to adequately assess the risk mitigation in the software.

Integration testing continues with the integration of CSCIs to create a software system. As the level of integration increases, testers will integrate the CSCIs with the rest of the operating environment, including the operating system and middleware on the host platform. To this point, much of the testing executes on workstations that provide testers with greater flexibility in the execution and monitoring of test parameters and extraction of data. However, test stations can introduce their own influences on the software system. Therefore, an essential part of testing is the integration with the host platform and the associated software environment.

Integration testing continues in steps until the entire software system is functioning on the target hardware platform. With each step in the integration process, testers will address different functionality and requirements in the tests. However, anomalies may arise in other portions of the software caused by the integration of seemingly unrelated software modules or CSCIs.

Throughout the integration testing process, the test organization will use simulators, stimulators, emulators, and other models to provide the necessary interfaces to the software and the system under test. Validation of these models is essential to the testing process. Invalid models will invalidate the testing. From a safety perspective, validation must include assurance that those aspects of the model required for the safety-significant testing function as closely as practical to the actual component being modeled.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

4.4.2.14 Regression Testing

Figure 4-45 illustrates the tasks associated with regression testing for the software safety team. Regression tests are a subset of other tests run on the system software at various levels of testing. Regression testing ensures that modifications to the software do not adversely affect functionality or safety. Regression testing can occur at any level. However, at the unit level, testers will often re-run all of the unit-level tests. At the integration test level, re-running all of the tests is often prohibitively expensive, so the testers will identify a group of tests that target specific functionality, often related to the modifications made, to verify that the modified software corrects the deficiency and retains the desired functionality. These regression tests may include a standard suite of tests or may be specifically developed to address a particular modification. Regression testing that occurs at the system integration level often uses a suite of tests that verify that the software retains the primary functionality and specific tests to verify correction of the anomaly or deficiency. Best practice dictates that regression testing should occur at all levels below the level at which the test team uncovered the anomaly, as well as at the level where the anomaly occurred. However as test schedules become compressed and resources become limited, test teams may only want to do regression testing at the unit level and the level during which the anomaly was discovered. The safety team must judge the adequacy of this decision based upon the criticality and complexity of the software and its mishap and hazard implications.

The safety team must identify those tests necessary to verify that any changes made to the unit or module do not adversely affect its safe functioning. If a SoS SHA was performed, then the functions that affect or are affected by SoS data (should have been “tagged” as such) must be tested at the system-to-SoS level to ensure that triggers and timing, as well as “right data,” are maintained. Without this SoS tagging, all changes are the same at a unit level. This requires that the team assess the proposed changes, including the proposed implementation, and provide any recommendations to those responsible for the change. As part of the recommendations, the safety team should identify the tests that should be re-run as either stand alone tests or as part of a regression test series. These safety-specific regression tests may occur at all levels, from unit to system integration, depending on the criticality of the software and the changes involved. At a minimum, the safety-specific regression tests should verify that safety-significant functionality is not adversely impacted by the changes and that no new safety-significant anomalies occur.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

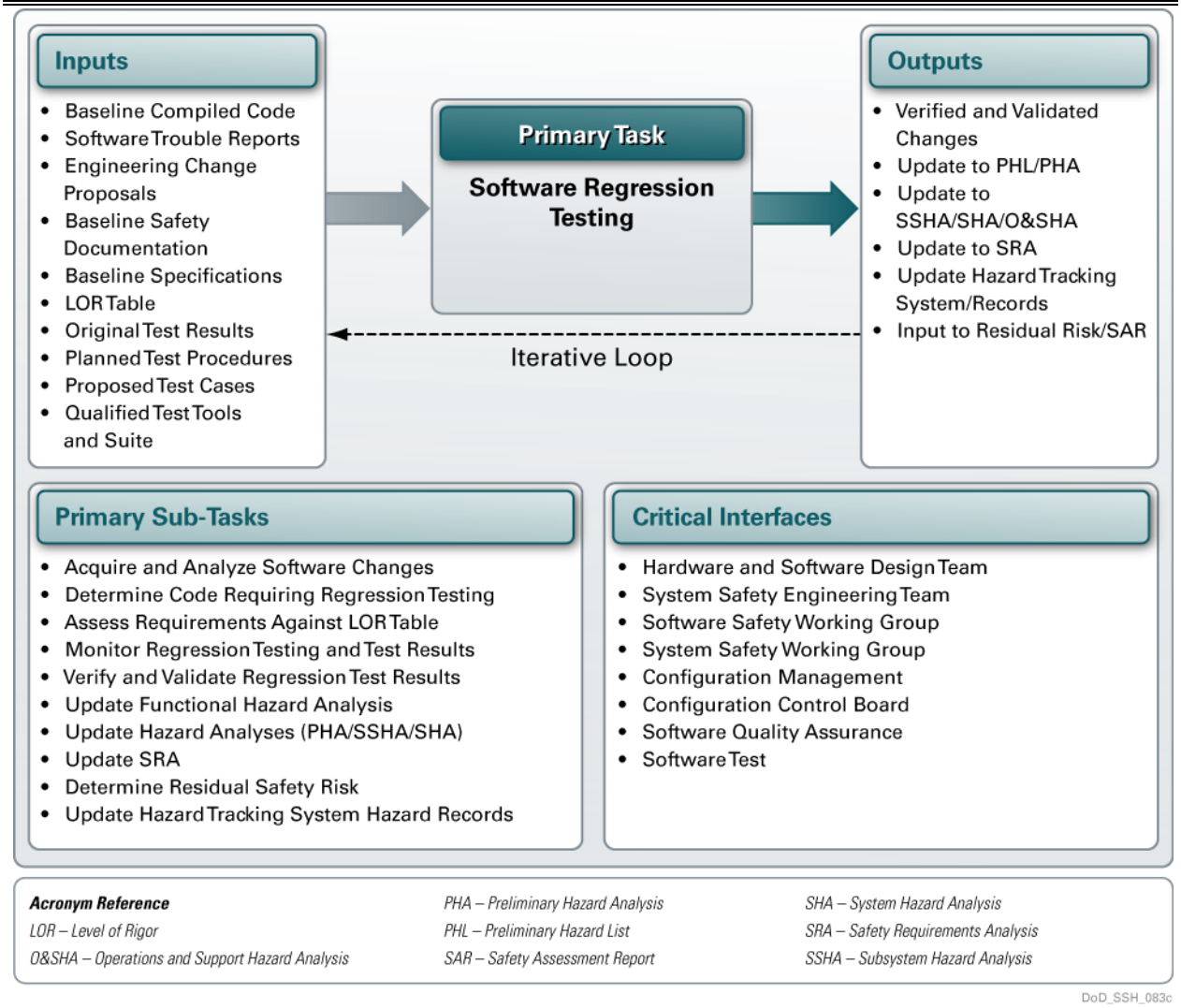

Figure 4-45: Software Regression Testing

4.4.3 Software Standards and Criteria Assessment

This section provides guidance to the SSS team to verify that software is developed in accordance with applicable safety-related standards and criteria. The software standards and criteria assessment will help define and establish the applicable GSSRs and identify the recommended approach to verify these requirements. The assessment (Figure 4-46) begins early in the development process as design requirements are tailored and implemented into system-level and top-tier software specifications, and continues through peer reviews and the analysis of test results and various reports from other IPTs. The assessment ultimately becomes an integral part of the overall SAR.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

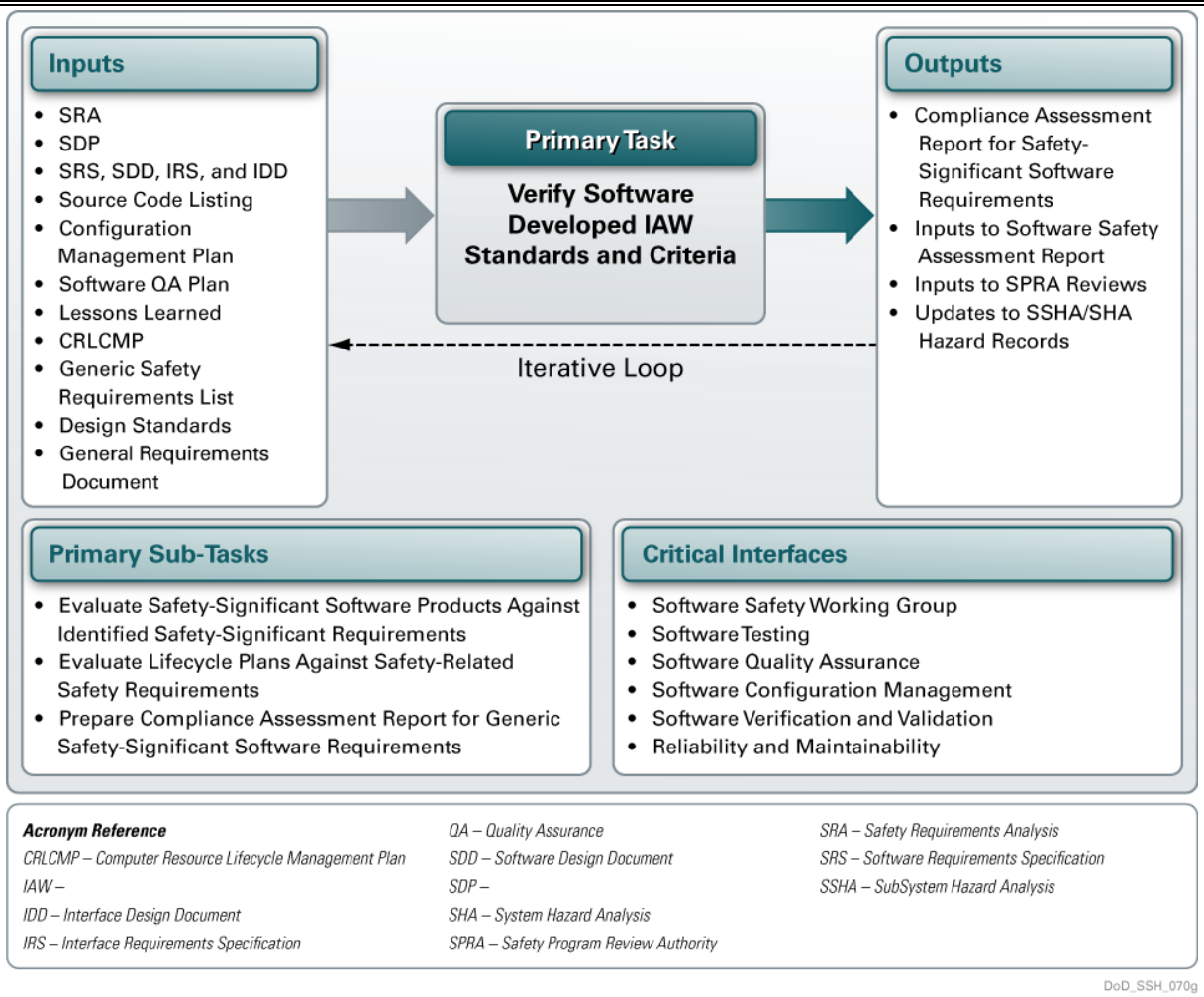

Figure 4-46: Software Requirements Verification

Standards and criteria include those extracted from best practice documents such as STANAG 4404 and military, Federal, and industry standards and handbooks; lessons learned; safety programs on similar systems; internal company documents; and other sources. The SSS team can delegate verification that the software is developed in accordance with syntactic restrictions and applicable software engineering standards and criteria to the SQA, SCM, and software testing (including the V&V) teams. This includes many of the generic software engineering requirements from STANAG 4404, Institute of Electrical and Electronic Engineers (IEEE) Standard 1498, and other related documents. Keep in mind that as existing guideline documents become dated, the SSS team must continue to identify new GSSRs from more current sources. An example would be the evolving safety-critical guidelines for JAVA.

The SQA and software configuration management processes include these requirements as a routine part of the normal compliance assessment process. Software developers and testers will test generic or system-specific safety test requirements as a normal part of the software testing

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

process. System safety staff must review test cases and test procedures and make recommendations for additional or modified procedures and tests to ensure complete coverage of applicable requirements. However, review of test cases and procedures alone may not be sufficient. A number of the generic requirements call for the safety analyst to ensure that the code meets both the intent and the letter of the safety requirement. As noted earlier, specifications and safety requirements may be interpreted differently by the software developer and may adequately meet the intent of the requirement. In some instances, this requires verification through peer reviews and an examination of the source code (see Section 4.3.7).

The generic software development requirements and guidelines are provided to the SQA team for incorporation into the assessment process, plans, and reviews. Although many of the specific test requirements may be assigned to individual software development teams for unit and integration testing, the software testing team generally assumes responsibility for incorporating generic test requirements into the test planning process. The Configuration Management team is responsible for the integration of requirements related to CM into the plans and processes that include participation by safety personnel. The latter includes safety staff participation in the CM process.

The SSS team reviews the assessment performed by the SQA team, incorporating the results for the safety-significant criteria into the final safety assessment. This assessment should occur on a real-time basis using representatives from the SSS team (or a member of the SQA team assigned responsibility for software safety). These representatives should participate in code reviews, walk-through processes, peer reviews, and the review of the software development process. The assessment should include the degrees of compliance or the rationale for non-compliance with each criterion.

Software safety analysts participate on a real-time basis with the CM process. Therefore, the assessment against applicable criteria can occur during this process. To a large extent, the degree of compliance depends on the degree of system safety involvement in the CM process and the thoroughness in fulfilling their roles.

The Compliance Assessment Report is a compilation of the compliance assessments noted above with a final assessment as to whether or not the system satisfies applicable safety requirements. The compliance assessment is a portion of the overall safety assessment process used to ascertain the residual risk associated with the system.

4.4.4 Safety Residual Risk Assessment

The inclusion of software adds complexity to the system safety residual risk assessment. The safety risk assessment is predicated on the severity of mishaps or hazards and the probabilities, qualitative or quantitative, coupled with the remaining conditions required, resulting in a hazardous condition or an accident. Engineering judgment and experience with similar systems provide the basis for qualitative probabilities, while statistical measurements (such as reliability predictions) provide the basis for quantitative probabilities. The functional contribution of

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

software as mishap/hazard causal factors is a significant contribution to the probability evaluation. The safety analyst uses this information to compile the probability of a mishap/ hazard occurring over the life of the system (the residual risk associated with the system). This information is coupled with estimates of the likelihood of satisfying the remaining conditions that result in an accident or mishap, resulting in an estimate of the risk.

Because the quantitative probability of software's causal contribution to a mishap/hazard is difficult to calculate, qualitative judgments are most common. Qualitative risk assessments must be applied based on an assessment by the analyst that sufficient analysis and testing have been performed through the LOR process. Sufficient analysis is required to identify the hazards, develop and incorporate safety requirements into the design, and analyze SSR implementation (including sufficient testing and analysis of test results) to provide a reasonable degree of assurance that the software will possess a sufficiently low level of safety risk. The software safety assessment begins early in the system development process with the compliance assessment of the safety requirements. However, the assessment cannot be completed until system-level testing in an operational environment is complete. This includes operational test and evaluation and the analyses that conclude that all identified hazards have been resolved. The software safety assessment process depicted in Figure 4-47 is generally complete when it is integrated with the SAR.

As described in Section 4.3.3, the analyst identifies the software safety-significant functions early in the analytical phase and assigns a mishap severity and software control category to each. The result is an SCI for that function and any mishap/hazard causal factors that reside within that function. The SCI provides an indication of the degree of assurance and integrity required via the LOR to ensure that it will execute safely in the system context. The SCI defines guidance for the amount of analysis and testing required to verify and validate the software associated with that function or causal factor. The SCI does not change unless the design is modified to reduce the degree of control that the software exercises over the potentially hazardous function. Based on the SCI and the associated LOR, analysis and testing performed on the software help to reduce the actual mishap probability in the system application. In this manner, a qualitative engineering judgment is used to assess mishap probability where software causal factors exist. Engineering judgment must always be reviewed by the SSWG so that consistent judgment with respect to stakeholder risk, individually and aggregated, is managed. The SSS team needs to document software causal factors, mitigations, SCI and LOR results, and associated hazards within the hazard tracking database.

As with any hazard analysis, closure of the hazard requires that the analyst review the results of the analyses performed against the tests conducted at both the component and system levels. Closure of hazards occurs as the design progresses. The SSS team analyzes the design and implementation of the safety significant functions, both safety critical and safety related, and determines whether they meet the intent of the SSR. The team also determines whether the implementation (hardware and software) provides sufficient interlocks, checks, and balances to ensure that the function will not contribute to a hazardous condition. Coupled with the results of testing performed on these functions, the analyst uses their best judgment as to whether the risk is sufficiently mitigated, records this information in the database, and presents this information to the SSWG for review and closure.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

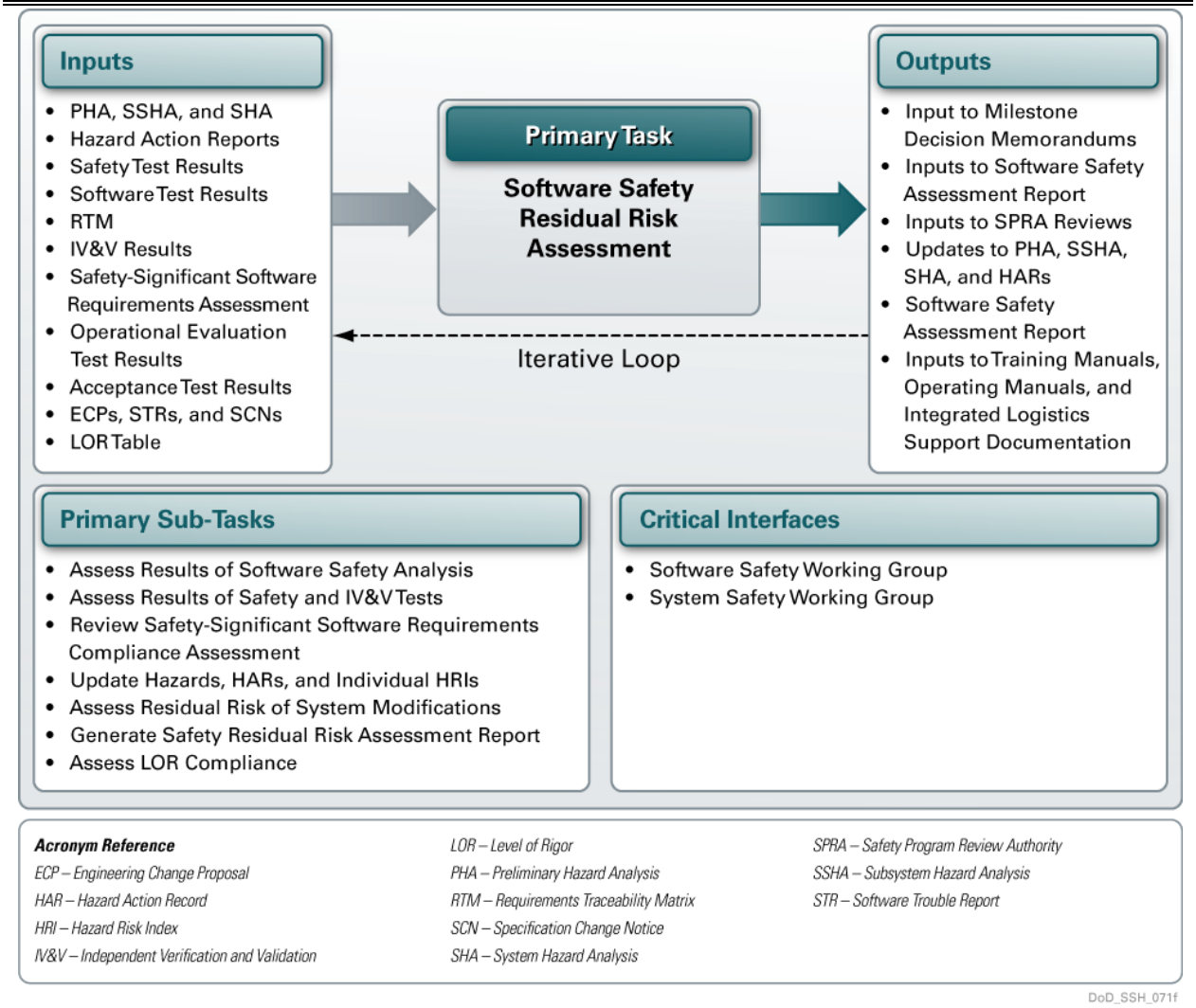

Figure 4-47: Software's Contribution to Residual Safety Risk Assessment

In performing the safety verification assessment and validation testing, the software safety analyst must examine a variety of metrics associated with testing. These include hazard data used in test cases, path coverage through safety modules, overall test coverage of the program, and usage-based testing. Testing that is limited to the usage base, to the changed portions of software only, or without actual hardware-in-the-loop is inadequate for safety. Safety-critical modules are often those that only execute when hazardous processing is initiated. This results in a low predicted usage, and consequently, usage-based testing provides little insight on those functions. The result is that anomalies may be present and undetected. The SSS team should assess the degree of test coverage and determine its adequacy. If the software testers are aware of the need for additional test coverage of safety-critical functions, they will be incorporated into the routine testing.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The safety program must subject new requirements identified during the analysis and testing phases to the same level of rigor as those in the original design. However, the SSS team must pay particular attention to these areas since they are the areas most likely to contain errors in the latter stages of development. This is a function of introducing requirements late in the process and reducing the time available for analysis and testing, which reduces the amount of re-engineering of associated functions and introduces unknowns. The potential interactions with other portions of the system interfaces will be unknown and will not receive the same degree of attention (especially at the integration testing level and higher) as the original requirements.

The SSS team must keep in mind the ultimate definition of acceptable risk as defined by the customer and the stakeholder(s). Where unacceptable or undesirable risks are identified, the SSS team, in coordination with the SSWG, must provide the rationale for recommending acceptance of that risk to the customer and the safety review authority. Even for systems that comply with the level of risk defined by customer requirements, the rationale for that assessment and the supporting data must be provided. This material is also documented in the SAR.

4.5 Safety Assessment Report

The SAR is generally a CDRL item for the safety analysis performed on a system. The purpose of the report is to provide management with an overall assessment of the risk associated with the system, including the software executing within the system context of an operational environment. This is accomplished by providing detailed analysis and testing evidence that all of the system hazards, including software-significant hazards, have been identified and eliminated, mitigated, or controlled to levels acceptable to the AE/PEO, PM, and PFS/Safety Manager. This assessment report be must include all of the analysis performed as a result of the recommendations provided in the previous sections.

The SAR contains a summary of the analyses performed and the results, the tests conducted and the results, and the compliance assessment. Section 4.5.1 is a sample outline for the SAR. Information within the SAR needs to encompass:

- The safety criteria and methodology used to classify and rank software-significant hazards (causal factors), including any assumptions made from which the criteria and methodologies were derived
- The results of the analyses and testing performed
- The hazards that have an identified residual risk and the assessment of that risk
- The list of significant hazards and the specific safety recommendations or precautions required to reduce safety risk
- A discussion of the engineering decisions made that affect the residual risk at a system level.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The conclusion of the SAR should be a statement by the PFS/Safety Manager describing the overall risk associated with the software in the system context and their acceptance of that risk. Section 4.5.1 identifies the contents of the SAR.

4.5.1 SAR Table of Contents

1. ***Introduction (Purpose, Scope, and Background)***

2. ***System Overview and Concept of Operations***

   - Provide a high-level hardware architecture overview
   - Describe system, subsystem, and interface functionality
   - Provide a detailed software architecture description
   - Describe the architecture, processors, and coding language
   - Describe CSCIs, CSUs, and functional interfaces
   - Identify safety-critical functionality and data.

3. ***System Description***

4. ***Hazard Analysis Methodology***

   - Describe the hazard-based approach and the structure of the hazard tracking database
   - Describe the SSR identification process (initial RTM development)
   - Define the hazard assessment approach (RAC and SCI matrices)
   - Define the tools, methods, and techniques used in the software safety analyses
   - Include a table with PHA-level hazards and RACs
   - Identify safety-critical functions that are software significant and indicate the hazards that are affected.

5. ***Document Hazard Analysis Results***

   - Provide detailed records of system-level hazards
   - Provide detailed records of subsystem-level hazards
   - Provide in-depth evidence of hazard causes to the level required to mitigate or control the hazards effectively, efficiently, and economically
   - Identify hazards that have software influence or causes (software causal factors in the functional, physical, or process context of the hazard).

6. ***Identify Hazard Elimination or Mitigation and Control Requirements***

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Describe sources of SSRs (e.g., System Design Reviews (SDRs), generics, functionally derived, and hazard control)
- Identify the initial SSRs for the system
- Provide evidence of SSR traceability (RTM and hazard tracking database updates)
- Identify functionally-derived SSRs based on detailed hazard cause analysis.

7. Provide Evidence of SSR Implementation in Design

- Describe the SSR verification process
- Describe SSR analysis, testing, and test results analysis
- Provide evidence of SSR verification (RTM and hazard tracking database updates).

8. Provide a Final Software Safety Assessment

- Provide an assessment of risk associated with each hazard in regard to software
- Identify any remaining open issues or concerns.

9. Appendices

- FHA, SSHA, or SHA (if not a separate deliverable)
- SRA (include RTM, SSR verification trees, and SSR test results)
- Hazard tracking database worksheets
- Any fault tree Analysis reports generated to identify software effects on hazards.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

APPENDIX A DEFINITION OF TERMS

A.1 Acronyms

ACAT	-	Acquisition Category
AE	-	Acquisition Executive
AFTI	-	Advanced Fighter Technology Integration
AIS	-	Automated Information System
ANSI	-	American National Standards Institute
ARP	-	Aerospace Recommended Practice
ARTE	-	Ada Runtime Environment
CAE	-	Component Acquisition Executive
CASE	-	Computer-Aided Software Engineering
CCB	-	Configuration Control Board
CCS	-	Command and Control System
CDD	-	Capabilities Development Document
CDR	-	Critical Design Review
CDRL	-	Contract Data Requirements List
CHI	-	Computer/Human Interface
CI	-	Configuration Item
CM	-	Configuration Management
CONOPS	-	Concept of Operations
COTS	-	Commercial-Off-The-Shelf
CPU	-	Central Processing Unit
CRC	-	Cyclic Redundancy Check
CRISD	-	Computer Resource Integrated Support Document
CRWG	-	Computer Resource Working Group
CSC	-	Computer Software Component
CSCI	-	Computer Software Configuration Item
CSR	-	Component Safety Requirement
CSSR	-	Contributing Software Safety-Requirement
CSU	-	Computer Software Unit
CTA	-	Critical Task Analysis
DA	-	Developing Agency
DAL	-	Development Assurance Level
DEF(AUST)	-	Australian Defense Standard
DEF-STAN	-	United Kingdom Defence Standard
DFD	-	Data Flow Diagram
DID	-	Data Item Description
DoD	-	Department of Defense
DoDD	-	Department of Defense Directive
DoDI	-	Department of Defense Instruction
DOD-STD	-	Department of Defense Standard

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

DSMC	-	Defense Systems Management College
ECP	-	Engineering Change Proposal
E/E/PES	-	Electrical/Electronic/Programmable Electronic Systems
EIA	-	Electronic Industries Association
E.O.	-	Executive Order
ESOH	-	Environment, Safety, and Occupational Health
FAA	-	Federal Aviation Administration
FCA	-	Functional Configuration Audit
FDA	-	U.S. Food and Drug Administration
FFD	-	Functional Flow Diagram
FHA	-	Functional Hazard Analysis
FMEA	-	Failure Modes and Effects Analysis
FOCC	-	Forward Operations Command Center
FTA	-	Fault Tree Analysis
GEIA	-	Government Electronics and Information Technology Association
GOTS	-	Government Off-The-Shelf
GSSR	-	Generic Software Safety Requirement
HFE	-	Human Factors Engineering
HHA	-	Health Hazard Assessment
HM	-	Hazardous Materials
HMI	-	Human/Machine Interface
HSI	-	Human Systems Integration
ICD	-	Initial Capabilities Document
ICWG	-	Interface Control Working Group
IDE	-	Integrated Development Environment
IDS	-	Interface Design Specification
IEC	-	International Electrotechnical Commission
IEEE	-	Institute of Electrical and Electronic Engineering
IER	-	Interface Exchange Requirement
ILS	-	Integrated Logistics Support
INCOSE	-	International Council on Systems Engineering
I/O	-	Input/Output
IPD	-	Integrated Product Development
IPT	-	Integrated Product Team
ISO	-	International Organization for Standardization
ISP	-	Information Support Plan
ITAA	-	Information Technology Association of America
IV&V	-	Independent Verification and Validation
JSSSEH	-	Joint Software Systems Safety Engineering Handbook

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

LOR	-	Level of Rigor
LOT	-	Level of Trust
MA	-	Managing Authority
MAA	-	Mission Area Analysis
MAIS	-	Major Automated Information System
MC/DC	-	Modified Condition/Decision Coverage
MDA	-	Milestone Decision Authority
MDAP	-	Major Defense Acquisition Program
MIL-STD	-	Military Standard
MS	-	Milestone
MSSR	-	Mitigating Software Safety Requirement
NASA	-	National Aeronautics and Space Administration
NASA GB	-	National Aeronautics and Space Administration Guide Book
NDI	-	Non-Developmental Item
NEPA	-	National Environmental Policy Act
NIST	-	National Institute for Standards and Technology
O&SHA	-	Operations and Support Hazard Analysis
OOA&D	-	Object-Oriented Analysis and Design
OS	-	Operating System
PA	-	Procuring Authority
PC	-	Program Counter
PCA	-	Physical Configuration Audit
PDL	-	Program Design Language
PDR	-	Preliminary Design Review
PEO	-	Program Executive Officer
PESHE	-	Programmatic Environment, Safety, and Occupational Health Evaluation
PFD	-	Process Flow Diagram
PFS	-	Principal For Safety
PHA	-	Preliminary Hazard Analysis
PHL	-	Preliminary Hazard List
PM	-	Program Manager
PMR	-	Program Management Review
POA&M	-	Plan of Actions and Milestones
POC	-	Point of Contact
PTR	-	Program Trouble Report
QA	-	Quality Assurance
QAP	-	Quality Assurance Plan
RAC	-	Risk Assessment Code
RAM	-	Risk Assessment Matrix

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

REP	-	Reliability Engineering Plan
RFP	-	Request for Proposal
RMP	-	Risk Management Plan
ROM	-	Read Only Memory
RTCA DO	-	RTCA, Inc.
RTM	-	Requirements Traceability Matrix
SAE ARP	-	Society of Automotive Engineers Aerospace Recommended Practice
SAF	-	Software Analysis Folder
SAR	-	Safety Assessment Report
SCC	-	Software Control Category
SCCSF	-	Safety-Critical Computing System Function
SCF	-	Safety-Critical Function
SCFL	-	Safety-Critical Functions List
SCI	-	Software Criticality Index
SCL	-	Software Criticality Level
SCM	-	Software Configuration Management
SDL	-	Safety Data Library
SDP	-	Software Development Plan
SDR	-	System Design Review
SEDS	-	System Engineering Detailed Schedule
SEE	-	Software Engineering Environment
SEMP	-	System Engineering Master Plan
SEMS	-	System Engineering Master Schedule
SEP	-	System Engineering Process
SHA	-	System Hazard Analysis
SIF	-	Safety Infrastructure Function
SIL	-	Safety Integrity Level
SIP	-	Software Installation Plan
SON	-	Statement of Need
SOO	-	Statement of Objectives
SoS	-	System of Systems
SOW	-	Statement of Work
SQA	-	Software Quality Assurance
SRA	-	Safety Requirements Analysis
SRB	-	Safety Review Board
SRS	-	Software Requirements Specifications
SSA	-	System Safety Assessment
SSF	-	Safety-Significant Function
SSG	-	System Safety Group
SSHA	-	Subsystem Hazard Analysis
SSMP	-	System Safety Management Plan
SSP	-	System Safety Program
SSPP	-	System Safety Program Plan
SSR	-	Software Safety Requirements
SSS	-	Software System Safety

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

SSWG	-	System Safety Working Group
STANAG	-	North Atlantic Treaty Organization Standardization Agreement
STP	-	Software Test Plan
STR	-	Software Trouble Report
STSC	-	Software Technology Support Center
SwSPP	-	Software Safety Program Plan
SwSSP	-	Software System Safety Program
SwSSWG	-	Software System Safety Working Group
TDP	-	Technical Data Package
TEMP	-	Test and Evaluation Master Plan
TLM	-	Top-Level Mishap
TRR	-	Test Readiness Review
TWG	-	Test Working Group
UK	-	United Kingdom
UML	-	Unified Modeling Language
UMS	-	Unmanned System
UMS WG	-	Unmanned Systems Working Group
V&V	-	Verification and Validation
WBS	-	Work Breakdown Structure
WCS	-	Weapon Control System

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

A.2 Definitions

Differing definitions continue to impact the design, development, test, and safety communities within the Department of Defense, other Government agencies, and commercial industry. An attempt has been made to select the most appropriate definition that possesses the best interpretation across the industry from a wide variety of sources. Where existing definitions appeared weak or outdated, a new definition was developed and labeled as "Proposed." While some may disagree with all or some aspects of the definition, the definitions herein were selected because of their common use or specific use on DoD programs.

Abort. The premature termination of a function, procedure, or mission. [Proposed]

Acceptable Risk. Part of identified risk that is allowed to persist without further engineering or management action to mitigate or control. [Proposed]

Acceptance. An action by an authorized representative of the acquirer by which the acquirer assumes ownership of software products as partial or complete performance of a contract.

Acceptance Criteria. The criteria that a system or component must satisfy in order to be accepted by a user, customer, or other authorized entity. See also: "Requirement; Test Criteria." [IEEE 610.12-1990]

Accident. Any unplanned event or series of events that results in death, injury, or illness to personnel or damage to or loss of equipment or property. Accident is synonymous with mishap. [FAA System Safety Handbook]

Acquirer. Stakeholder that acquires or procures a product or service from a supplier. NOTE: The acquirer could be one of the following—buyer, customer, owner, or purchaser. [ISO/ International Electrotechnical Commission (IEC) 12207:2008(E)]

Acquiring Agency. An acquiring agency may or may not produce software; it could procure. [Proposed]

Anomalous Behavior. System or software behavior which is not in accordance with the documented specifications or requirements or is the result of incorrect requirements. [Proposed]

Anomaly. A state or condition which is not expected. An anomaly may or may not be hazardous, but it is the result of a transient hardware, bad requirement, or coding error. [Proposed]

Architecture. The organizational structure of a system or component. [IEEE 610.12 - 1990]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Assembly. A number of parts, subassemblies, or any combination thereof joined together to perform a specific function and which can be disassembled without destruction of the designated use. [SAE ARP 4761]

Audit. Independent assessment of software products and processes conducted by an authorized person in order to assess compliance with requirements. [ISO/IEC 12207:2008(E)]

Authorized Entity. An individual operator or control element authorized to direct or control the system or SoS functions or mission. [Proposed]

Autonomous. (1) Operations of an unmanned system wherein the UMS receives its mission from the human operator and accomplishes that mission with or without further human-to-machine interaction. (2) System responds to stimulus/stimuli in a self-contained manner independent of human interactions. [Proposed] NOTE: The level of human-to-machine interaction, mission complexity, and environmental difficulty determine the level of autonomy. Autonomy level designations can also be applied to tasks lower in scope than mission tasks.

Baseline. Specification or product that has been formally reviewed and agreed upon that thereafter serves as the basis for further development and that can be changed only through formal change management procedures. [ISO/IEC 12207:2008(E)]

Behavioral Design. The design of how an overall system or CSCI will behave, from a user's point of view, in meeting requirements, ignoring the internal implementation of the system or CSCI. This design contrasts with architectural design, which identifies the internal components of the system or CSCI, and with the detailed design of those components.

Build. The period of time during which a version of software is developed.

Built-in-Test. Equipment or software embedded in operational components or systems, as opposed to external support units, which perform a test or sequence of tests to verify mechanical or electrical continuity of hardware, or the proper automatic sequencing, data processing, and readout of hardware or software systems. [National Institute for Standards and Technology (NIST) Special Publication 1011]

Cascading Failure. A failure of which the probability of occurrence is substantially increased by the existence of a previous failure. [SAE ARP 4754]

Causal Factors. (1) The particular and unique set of circumstances that can contribute to a hazard. (2) The combined hazard sources and initiating mechanisms that may be the direct result of a combination of failures, malfunctions, external events, environmental effects, errors, inadequate design, or poor judgment. [Proposed]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Commercial Off-the-Shelf Software. (1) An item that is commercially available, leased, licensed, or sold to the general public which requires no special modification or maintenance over its intended lifecycle. (2) Systems which are commercially manufactured and then tailored for specific uses. This is most often used in military, computer, and robotic systems. COTS systems are in contrast to systems that are produced entirely and uniquely for the specific application. [Proposed]

Common Cause Failure. An event or failure which bypasses or invalidates redundancy or system independence with the potential of causing a mishap. [Proposed]

Complexity. An attribute of systems or items which makes their operation difficult to comprehend. Increased system complexity is often caused by sophisticated components and multiple interrelationships. [SAE ARP 4761]

Computer Firmware. The combination of a hardware device and instructions of computer data that reside as read-only software on the hardware device. The software cannot be readily modified under program control. [Defense Acquisition University Glossary 11th Edition]

Computer Program. A combination of computer instructions and data definitions that enable computer hardware to perform computational or control functions. (See also: "Software.") [IEEE 610.12-1990]

Computer Software Component. A functionally or logically distinct part of a computer software configuration item, typically an aggregate of two or more software units. [IEEE 610.12-1990]

Computer Software Configuration Item. An aggregation of software that is designated for configuration management and treated as a single entity in the configuration management process. [IEEE 610.12-1990]

Computer Software Unit. The smallest subdivision of a CSCI for the purposes of functional and engineering management. CSUs are typically separately compiled and testable units of code. [Proposed]

Concept of Operations. A verbal or graphic statement, in broad outline of a Commander's assumption or intent, in regard to an operation or series of operations. CONOPS are frequently embodied in campaign plans and operation plans; in the latter case, this is particularly true when the plans cover a series of connected operations to be carried out simultaneously or in succession. [DoD Joint Publication 1-02]

Concurrent Operations. Operations performed simultaneously and in close enough proximity that an incident with one operation could adversely influence the other. [Department of Energy Manual 4401]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Condition. An existing or potential state such as exposure to harm, toxicity, energy source, activity, etc. [MIL-STD 882]

Configuration Item. An aggregation of hardware, software, or both that is designated for configuration management and treated as a single entity in the configuration management process. [IEEE 610.12-1990]

Contractor. A private sector enterprise or the organizational element of the Department of Defense or any other Government agency engaged to provide services or products within agreed limits specified by the MA. [MIL-STD 882]

Contributing Software Safety Requirements. A subcategory of the defined software safety requirements of a program. CSSRs are requirements contained within the software specifications that contribute to the safety risk of the system by the functionality that they will perform (e.g., "arm the missile" or "fire the missile"). [Proposed]

Control Program. The inherent set of control instructions which defines the capabilities, actions, and responses of the system. This program is usually not intended to be modified by the user. [American National Standards Institute (ANSI)/Robot Safety Standard 15.06]

Cooperative Operations. The ability of two or more systems to share data, coordinate maneuvers, and synergistically perform tasks. [Proposed]

Criticality. A measure of the impact of a failure mode on the mission objective. Criticality combines the potential frequency of the occurrence and the level of severity of the failure mode. [Proposed]

Data Link. The means of connecting one location (or system) to another for the purpose of transmitting or receiving data. [DoD Joint Publication 1-02]

Data Type. A class of data characterized by the members of the class and operations that can be applied to them (e.g., integer, real, or logical). [IEEE 729-1983]

Deactivated Code. A software program, routine, or set of routines which were specified, developed, and tested for one system configuration are disabled for the existing system or new system configuration. The disabled functions may or may not be fully tested in the new system configuration to demonstrate that an inadvertent activation of the function would result in a safe outcome within the intended environment. [Proposed]

Dead Code. (1) Dead code is code unintentionally included in the baseline, left in the system from an original software configuration that is not erased or overwritten, or code not tested for the current configuration and environment. (2) Executable code (or data) which, as a result of design, maintenance, or installation error, cannot be executed (code) or used (data) in any

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

operational configuration of the system and is not traceable to a requirement. [National Aeronautics and Space Administration Guidebook (NASA-GB) 8719.13]

Defect. A condition or characteristic in any hardware or software which is not in compliance with the specified configuration, or inadequate or erroneous configuration identification which has resulted in or may result in configuration items (CIs) that do not fulfill approved operational requirements. A defect may exist without leading to a failure. [Proposed]

Deliverable Software Product. A software product that is required by the contract to be delivered to the acquirer or other designated recipient.

Derived Requirements. Additional requirements resulting from design or implementation decisions during the development process. Derived requirements are not directly traceable to higher level requirements, though derived requirements can influence higher level requirements. [SAE ARP 4761]

Design. The process of defining the architecture, components, interfaces, and other characteristics of a system or component. [IEEE 610.12-1990]

Environment. (1) The aggregate of the external procedures, conditions, and objects that affect the development, operation, and maintenance of a system. (2) Everything external to a system which can affect or be affected by the system. [NASA-GB-8719.13]

Error. (1) The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. For example, a difference of 30 meters between a computed result and the correct result. (2) An incorrect step, process, or data definition. For example, an incorrect instruction in a computer program. (3) An incorrect result. For example, a computed result of 12 when the correct result is 10. (4) A human action that produces an incorrect result. For example, an incorrect action on the part of a programmer or operator. [IEEE 610.12-1990]

Fail Safe. A design feature that ensures that the system remains safe, or in the event of a failure, will cause the system to revert to a state which will not cause a mishap. [MIL-STD 882]

Failure. The inability of an item to perform its intended function. [SAE ARP 4754]

Failure Tolerance. The ability of a system or subsystem to perform its function(s) or maintain control of a hazard in the presence of failures within its hardware, firmware, or software. [NASA-GB-8719-13]

Fault. Any change in state of an item that is considered to be anomalous and may warrant some type of corrective action. Examples of faults included device errors reported by Built-In Test/Built-In Test Equipment; out-of-limits conditions on sensor values; loss of communication

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

with devices; loss of power to a device; communication error on bus transaction; software exceptions (e.g., divide by zero and file not found); rejected commands; measured performance values outside of commanded or expected values; an incorrect step, process, or data definition in a computer program; etc. Faults are preliminary indications that a failure may have occurred. [NASA-GB-8719-13]

Firmware. The combination of a hardware device and computer instructions and/or computer data that resides as read-only software on the hardware device. [IEEE 610.12-1990]

Formal Methods. (1) The use of formal logic, discrete mathematics, and machine-readable languages to specify and verify software. (2) The use of mathematical techniques in design and analysis of the system. [NASA-GB-8719-13]

Function. A task, action, or activity that must be performed to achieve a desired outcome. [International Council on Systems Engineering (INCOSE), Systems Engineering Handbook]

Fusion. The combining or blending of relevant data and information from single or multiple sources (sensors, logistics, etc.) into representational formats that are appropriate to support the interpretation of the data and information and to support system goals like recognition, tracking, situation assessment, sensor management, or system control. Fusion involves the processes of acquisition, filtering, correlation, integration, comparison, evaluation, and related activities to ensure proper correlations of data or information exist and draw out the significance of those correlations. [National Institute for Standards and Technology (NIST) Special Publication 1011]

Generic Software Safety Requirement. A subcategory of the defined software safety requirements of a program. GSSRs are a product of documented software development, system safety best practices, and lessons learned from legacy programs. [Proposed]

GOTS. Government-off-the-shelf (GOTS) technologies refer to Government-created software, usually from another project. The software was not created by the current developers. Usually source code and all available documentation, including test and analysis results, are included. [NASA-GB-8719-13]

Graceful Degradation. (1) A planned stepwise reduction of function(s) or performance as a result of failure, while maintaining essential function(s) and performance. (2) The ability to continue to operate with lesser capabilities in the face of faults or failures or when the number or size of tasks to be accomplished exceeds the capability to complete. [NASA-GB-8719-13]

Hardware. Physical equipment used to process, store, or transmit computer programs or data. [IEEE 610.12-1990]

Hardware Configuration Item. An aggregation of hardware that is designated for configuration management and treated as a single entity in the configuration management process. [IEEE 610.12-1990]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Hazard. A condition that is a prerequisite to a mishap. [MIL-STD 882]

Hazard Causal Factors. The particular or unique condition, set of circumstances, or initiating mechanism that contributes to the existence of a hazard. Causal factors may be the result of failures, malfunctions, external events, environmental effects, errors, poor design, or a combination thereof. Causal factors generally break down into categories of hardware, software, human action, procedures, and environmental factors. [Proposed]

Independence. (1) A design concept which ensures that the failure of one item does not cause a failure of another item. (2) Separation of responsibilities that ensures the accomplishment of objective evaluation. [SAE ARP 4761]

Independent Verification and Validation. Verification and validation performed by an organization that is technically, managerially, and financially independent of the development organization. [IEEE 610.12-1990]

Integrity. Attribute of a system or item indicating that it can be relied upon to work correctly on demand. [SAE ARP 4754]

Latent Failure. A failure which is not detected or annunciated when it occurs. [SAE ARP 4761]

Level of Authority. The degree to which an entity is invested with the power to access the control and functions of a UMS.

Level I – Reception and transmission of secondary imagery or data

Level II – Reception of imagery or data directly from the UMS

Level III – Control of the UMS payload

Level IV – Full control of the UMS, excluding deployment and recovery

Level V – Full control of the UMS, including deployment and recovery.

[Proposed Unmanned Systems Working Group (UMS WG)-3]

Level of Autonomy. Set(s) of progressive indices, typically given in numbers, identifying a unmanned system's capability for performing autonomous missions. Two types of metrics are used—Detailed Model for Autonomy Levels and Summary Model for Autonomy Levels. [NIST Special Publication 1011]

Level of Rigor. A specification of the depth and breadth of software analysis, test, and verification activities necessary to provide a sufficient level of confidence that a safety-critical or safety-related software function will perform as required.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Life Cycle. Evolution of a system, product, service, project, or other human-made entity from conception through retirement. [ISO/IEC 12207:2008(E)]

Life Cycle Model. Framework of processes and activities concerned with the system lifecycle that may be organized into stages. Also acts as a common reference for communication and understanding. [ISO/IEC 12207:2008(E)]

Malfunction. The occurrence of a condition whereby the operation is outside specified limits. [SAE ARP 4761]

Managing Activity. The organizational element of the Department of Defense assigned acquisition management responsibility for the system, or prime or associate contractors or subcontractors who impose system safety tasks on their suppliers. [MIL-STD 882]

Mishap. An unplanned event or series of events resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment. An accident. [MIL-STD 882]

Mishap Probability. The aggregate probability of occurrence of the individual events/hazards that might create a specific mishap. [MIL-STD 882]

Mishap Risk. An expression of the impact and possibility of a mishap in terms of potential mishap severity and probability of occurrence. [MIL-STD 882D]

Mishap Risk Assessment. The process of characterizing hazards within risk areas and critical technical processes, analyzing them for their potential mishap severity and probabilities of occurrence, and prioritizing them for risk mitigation actions.

Mishap Severity. An assessment of the consequences of the most reasonable credible mishap that could be caused by a specific hazard. [MIL-STD 882]

Mitigating Software Safety Requirement. A subcategory of the defined software safety requirements of a program. MSSRs are normally identified during in-depth mishap and hazard causal analysis and are derived for the purpose of mitigating or controlling failure pathways to the mishap or hazard. [Proposed]

Mode. Modes identify operational segments within the system lifecycle, generally defined in the CONOPS. Modes consist of one or more sub-modes. A system may be in only one mode, but may be in more than one sub-mode at any given time. [Proposed UMS WG-6]

Non-Development Item. A previously developed item of supply used without modification and where the design control agent of the item is not the system design agent for the application or system being acquired. [Proposed]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

N-Version Software. Software developed in two or more versions using different specifications, programmers, languages, platforms, compilers, or a combination of these. This is usually an attempt to achieve independence between redundant software items. Research has shown that this method usually does not achieve the desired reliability, and it is no longer recommended. [NASA-GB-8719.13]

Patch. A modification to a computer sub-program that is separately compiled and inserted into the machine code of a host or parent program. This avoids modifying the source code of the host or parent program. Consequently the parent or host source code no longer corresponds to the combined object code. [NASA-GB-8719-13]

Process. A sequence of steps performed for a given purpose; for example, the software development process. [IEEE 610.12-1990]

Qualification Testing. Testing conducted to determine whether a system or component is suitable for operational test. [IEEE 610.12-1990]

Quality. (1) The degree to which a system, component, or process meets specified requirements. (2) The degree to which a system, component, or process meets customer or user needs or expectations. [IEEE610.12-1990]

Quality Assurance. (1) A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements. (2) A set of activities designed to evaluate the process by which products are developed or manufactured. [IEEE610.12-1990]

Quality Metric. (1) A quantitative measure of the degree to which an item possesses a given quality attribute. (2) A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute. [IEEE610.12-1990]

Reengineering. The process of examining and altering an existing system to reconstitute it in a new form. May include reverse engineering (analyzing a system and producing a representation at a higher level of abstraction, such as design from code), restructuring (transforming a system from one representation to another at the same level of abstraction), redocumentation (analyzing a system and producing user or support documentation), forward engineering (using software products derived from an existing system, together with new requirements, to produce a new system), retargeting (transforming a system to install it on a different target system), and translation (transforming source code from one language to another, or from one version of a language to another).

Regression Testing. The testing of software to confirm that functions that were previously performed correctly continue to perform correctly after a change has been made. [NASA-GB-8719-13]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Requirement. (1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. (3) A documented representation of a condition or capability as in (1) or (2). [IEEE 610.12-1990]

Residual Mishap Risk. The remaining mishap risk that exists after all mitigation techniques have been implemented or exhausted, in accordance with the system safety design order of precedence. [MIL-STD 882]

Reusable. Pertaining to a software module or other work product that can be used in more than one computer program or software system.

Safe State. A state in which the system poses an acceptable level of risk for the operational mode. [Proposed UMS WG-6]

Safety. Freedom from those conditions that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment. [MIL-STD 882]

Safety-Critical. A term applied to a condition, event, operation, process, or item of whose mishap or hazard severity consequence is deemed to be either Catastrophic or Critical by definition (e.g., safety-critical function, safety-critical path, and safety-critical component). [Proposed]

Safety-Critical Computer Software Components. Those computer software components and units whose errors can result in a potential hazard, loss of predictability, or loss of control of a system. [MIL-STD 882]

Safety-Critical Function. A function whose failure to operate or incorrect operation will directly result in a mishap of either Catastrophic or Critical severity. [Proposed]

Safety-Related. A term applied to a condition, event, operation, process, or item whose mishap or hazard severity consequence is deemed to be less than Catastrophic or Critical by definition. [Proposed]

Safety-Related Function. A function whose failure to operate or incorrect operation will directly result in a mishap of a severity less than Catastrophic or Critical. [Proposed]

Safety Significant. A term applied to a condition, event, operation, process, or item that possesses a mishap or hazard severity consequence by definition. That which is defined as "safety significant" can either be safety critical or safety related. [Proposed]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Safety-Significant Function. A function whose failure to operate or incorrect operation will contribute to a mishap. [Proposed]

Semi-Autonomous. A mode of control of a system wherein the human operator plans a mission for the system, conducts the assigned mission, and requires infrequent human operator intervention when it needs further instructions. [Proposed]

Software Development. The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use. NOTE: These activities may overlap or be performed iteratively. [IEEE 610.12-1990]

Software Duration Test. Software testing that subjects the software to perform over the expected full life in which the software is required to perform correctly and no data senescence occurs. [Proposed]

Software Engineering. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; the application of engineering to software. [IEEE 610.12-1990]

Software Error. The difference between a computed, observed, or measured value or condition and the true, specified or theoretically correct value or condition. [NASA-GB-8719-13]

Software Fault. An incorrect step, process, or data definition in a computer system. [NASA-GB-8719-13]

Software Partitioning. Separation, physically and/or logically, of safety-significant functions from other non-safety-significant functionality. [Proposed]

Software Stress Test. Software testing that subjects the software to extreme external conditions and anomalous situations in which the software is required to perform correctly. [Proposed]

Stakeholder. Individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations. [ISO/IEC 12207:2008(E)]

Subsystem. An element of a system that in itself may constitute a system. [MIL-STD 882]

System. A composite, at any level of complexity, of personnel, procedures, materials, tools, equipment, facilities, and software. The elements of this composite entity are used together in the intended operational or support environment to perform a given task or achieve a specific purpose, support, or mission requirement. [MIL-STD 882]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

System Architecture. The arrangement of elements and subsystems and the allocation of functions to meet system requirements. [INCOSE Systems Engineering Handbook]

Systems-of-Systems. A collection or network of systems functioning together to achieve a common purpose. SoS are distinguished by five principal characteristics. NOTE: These five characteristics are useful in distinguishing very large and complex but monolithic systems from true systems-of-systems. Systems-of-systems possess:

1. Operational independence of the elements – If the system-of-systems is disassembled into its component systems, the component systems must be able to usefully operate independently. The system-of-systems is composed of systems which are independent and useful in their own right.
2. Managerial independence of the elements – The component systems can and do operate independently. The component systems are separately acquired and integrated but maintain a continuing operational existence independent of the system-of-systems.
3. Evolutionary development – The system-of-systems does not appear fully formed. Its development and existence is evolutionary, with functions and purposes added, removed, and modified with experience.
4. Emergent behavior – The system performs functions and carries out purposes that do not reside in any one component system. These behaviors are emergent properties of the system-of-systems. The principal purposes of the system-of-systems are fulfilled by these behaviors.
5. Geographic distribution – The geographic extent of the component systems is large. Large is a nebulous and relative concept as communication capabilities increase, but at a minimum it means that the components can readily exchange only information, not substantial quantities of mass or energy. [Proposed]

System Safety. The application of engineering and management principles, criteria, and techniques to optimize all aspects of safety within the constraints of operational effectiveness, time, and cost throughout all phases of the system lifecycle. [MIL-STD 882]

System Safety Engineer. An engineer who is qualified by training or experience to perform system safety engineering tasks. [MIL-STD 882]

System Safety Engineering. An engineering discipline requiring specialized professional knowledge and skills in applying scientific and engineering principles, criteria, and techniques to identify and eliminate hazards and reduce the associated risk. [MIL-STD 882]

System Safety Group/Working Group. A formally chartered group representing organizations initiated during the system acquisition program, and organized to assist the MA system Program Manager in achieving system safety objectives. Regulations of the DoD Components define requirements, responsibilities, and memberships. [MIL-STD 882]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

System Safety Management. A management discipline that defines system safety program requirements and ensures the planning, implementation, and accomplishment of system safety tasks and activities consistent with overall program requirements. [MIL-STD 882]

System Safety Manager. A person responsible to program management for setting up and managing the system safety program. [MIL-STD 882]

System Safety Program. The combined tasks and activities of system safety management and system safety engineering implemented by acquisition project managers. [MIL-STD 882]

System Safety Program Plan. A description of the planned tasks and activities to be used by the contractor to implement the required system safety program. This description includes organizational responsibilities, resources, methods of accomplishment, milestones, depth of effort, and integration with other program engineering and management activities and related systems. [MIL-STD 882]

System State. A condition in which a system or subsystem can be said to exist exclusively. A system or subsystem may be in only one state at a time. States are unique and may be binary (i.e., they are either true or not true). [Proposed UMS WG-6]

Technical Data Package. The engineering drawings, associated lists, process descriptions, and other documents that define system product and process physical geometry; material composition; performance characteristics; and manufacture, assembly, and acceptance test procedures. [INCOSE Systems Engineering Handbook]

Testability. Extent to which an object or feasible test can be designed to determine whether a requirement is met. [ISO/IEC 12207:2008(E)]

Test Case. A set of test inputs, execution conditions, and expected results used to determine whether the expected response is produced. [NASA-GB-8719-13]

Test Coverage. Extent to which the test cases test the requirements for the system or software product. [ISO/IEC 12207:2008(E)]

Test Procedure. (1) A specified way to perform a test. (2) Detailed instructions for the set-up and execution of a given set of test cases and instructions for the evaluation of results executing the test cases. [NASA-GB-8719-13]

Undocumented Code. Software code that is used by the system but is not documented in the software design. This usually pertains to COTS technology because the documentation is not always available. [NASA-GB-8719-13]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Validation. The determination that the requirements for a product are sufficiently correct and complete. [SAE ARP 4754]

Verification. The evaluation of an implementation of requirements to determine that they have been met. [SAE ARP 4754]

Version. Identified instance of an item. NOTE: Modification to a version of a software product, resulting in a new version, requires configuration management. [ISO/IEC 12207:2008(E)]

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

APPENDIX B REFERENCES

B.1 Government References

Allied Ordnance Publication 52, *Guidance on Software Safety Design and Assessment of Munitions-Related Computing System;* November 2008.

DoDD 5000.1, *Defense Acquisition*; March 15, 1996.

DoDI 5000.2R, *Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information Systems*; March 15, 1996.

Department of Defense Standard (DOD-STD) 2167A, *Military Standard Defense System Software Development*; February 29, 1988.

MIL-STD 882B, *System Safety Program Requirements*; March 30, 1984.

MIL-STD 882C, *System Safety Program Requirements*; January 19, 1993.

MIL-STD 882D, *Department of Defense Standard Practice, System Safety*; February 10, 2000.

MIL-STD 882D, Revision 1, (DRAFT), *Department of Defense Standard Practice, System Safety, Environment, Safety, and Occupational Health Risk Management Methodology for Systems Engineering*, April 1, 2010.

MIL-STD 498, *Software Development and Documentation*; December 5, 1994.

FAA Order 1810, *Acquisition Policy*; January 1998.

FAA Order 8000.70, *FAA System Safety Program*, December 30, 2001.

RTCA DO-178B, *Software Considerations In Airborne Systems And Equipment Certification;* December 1, 1992.

Commandant Instruction Manual 411502D, *System Acquisition Manual*; December 27, 1994.

NASA Safety Standard 1740.13, *Interim Software Safety Standard*, June 1994.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Department of the Air Force Software Technology Support Center (STSC), *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems, Command and Control Systems, Management Information Systems*; Version-2, June 1996, Volumes 1 and 2.

Air Force Inspection and Safety Center System Safety Handbook 1-1, *Software System Safety Handbook*; September 5, 1985.

B.2 Commercial References

Electronic Industries Association (EIA) 6B, G-48, Electronic Industries Association, *System Safety Engineering in Software Development*; 1990.

IEC 1508 - (Draft), International Electrotechnical Commission, *Functional Safety: Safety-Related System*; June 1995.

IEC 61508: International Electrotechnical Commission, *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*; December 1997.

IEEE STD 1228, Institute of Electrical and Electronics Engineers, Inc., *Standard for Software Safety Plans*; 1994.

IEEE STD 829, Institute of Electrical and Electronics Engineers, Inc., *Standard for Software Test Documentation*, 1983.

IEEE STD 830, Institute of Electrical and Electronics Engineers, Inc., *Guide to Software Requirements Specification*; 1984.

IEEE STD 1012, Institute of Electrical and Electronics Engineers, Inc., *Standard for Software Verification and Validation Plans*; 1987.

ISO 12207-1, *Information Technology-Software*; 1994.

Society of Automotive Engineers, Aerospace Recommended Practice 4754, *Certification Considerations for Highly Integrated or Complex Aircraft Systems*; November 1996.

Society of Automotive Engineers, Aerospace Recommended Practice 4761, *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*; December 1996.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

B.3 Individual References

Brown, Michael L., *Software Systems Safety and Human Error*, Proceedings: COMPASS, 1988.

Brown, Michael L., *What is Software Safety and Whose Fault Is It Anyway?* Proceedings: COMPASS, 1987.

Brown, Michael L., *Applications of Commercially Developed Software in Safety-Critical Systems*; Proceedings of Parari '99, November 1999.

Bozarth, John D., *Software Safety Requirement Derivation and Verification*; Hazard Prevention, Q1, 1998.

Bozarth, John D., *The MK 53 Decoy Launching System: A Hazard-Based Analysis Success*; Proceedings: Parari '99, Canberra, Australia.

Card, D.N. and Schultz, D.J., *Implementing a Software Safety Program*; Proceedings: COMPASS, 1987.

Church, Richard P., *Proving a Safe Software System Using a Software Object Model*; Proceedings: 15th International System Safety Society Conference, 1997.

Connolly, Brian, *Software Safety Goal Verification Using Fault Tree Techniques: A Critically Ill Patient Monitor Example*; Proceedings: COMPASS, 1989.

De Santo, Bob, *A Methodology for Analyzing Avionics Software Safety*; Proceedings: COMPASS, 1988.

Dunn, Robert and Ullman, Richard, *Quality Assurance for Computer Software*; McGraw Hill, 1982.

Ericson, C.A., *Anatomy of a Software Hazard*; Briefing Slides, Boeing Computer Services, June 1983.

Foley, Frank, *History and Lessons Learned on the Northrop-Grumman B-2 Software Safety Program*; Paper, Northrop-Grumman Military Aircraft Systems Division, 1996.

Forrest, Maurice and McGoldrick, Brendan, *Realistic Attributes of Various Software Safety Methodologies*; Proceedings: Ninth International System Safety Society, 1989.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Gill, Janet A., *Safety Analysis of Heterogeneous-Multiprocessor Control System Software*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, December 1990.

Hammer, William R., *Identifying Hazards in Weapon Systems – The Checklist Approach*; Proceedings: Parari '97, Canberra, Australia.

Kjos, Kathrin, *Development of an Expert System for System Safety Analysis*; Proceedings: Eighth International System Safety Conference, Volume II, date unknown.

Lawrence, J.D., *Design Factors for Safety-Critical Software*; NUREG/CR-6294, Lawrence Livermore National Laboratory, November 1994.

Lawrence, J.D., *Survey of Industry Methods for Producing Highly Reliable Software*; NUREG/CR-6278, Lawrence Livermore National Laboratory, November 1994.

Leveson, Nancy G., *SAFEWARE: System Safety and Computers, a Guide to Preventing Accidents and Losses Caused by Technology*; Addison Wesley, 1995.

Leveson, Nancy G., *Software Safety: Why, What, and How*; Computing Surveys, Volume 18, No. 2, June 1986.

Littlewood, Bev and Strigini, Lorenzo, *The Risks of Software*; Scientific American, November 1992.

Mattern, S.F., *Software Safety*; Masters Thesis, Webster University, St. Louis, MO, 1988.

Capt. Mattern, S.F., *Defining Software Requirements for Safety-Critical Functions;* Proceedings: 12th International System Safety Conference, 1994.

Mills, Harland D., *Engineering Discipline for Software Procurement*; Proceedings: COMPASS, 1987.

Moriarty, Brian and Roland, Harold E., *System Safety Engineering and Management*; Second Edition, John Wiley & Sons, 1990.

Russo, Leonard, *Identification, Integration, and Tracking of Software System Safety Requirements*; Proceedings: Twelfth International System Safety Conference, 1994.

Unknown Author, *Briefing on the Vertical Launch,* Anti-Submarine Rocket, Minutes, 2nd Software System Safety Working Group, March 1984.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

B.4 Other References

Australian Defense Standard (DEF(AUST)) 5679, Army Standardisation, *The Procurement of Computer-Based Safety Critical Systems*; May 1999.

United Kingdom (UK) Ministry of Defence. Interim Defence Standard 00-54, *Requirements for Safety-Related Electronic Hardware in Defence Equipment*; April 1999.

UK Ministry of Defence. Defence Standard 00-55, *Requirements for Safety-Related Software in Defence Equipment*; Issue 2, 1997.

UK Ministry of Defence. Defence Standard 00-56, *Safety Management Requirements for Defence Systems*; Issue 2, 1996.

International Electrotechnical Commission, IEC 61508, *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*; DRAFT 61508-2 Ed 1.0, 1998.

Document ID: CA38809-101, *International Standards Survey and Comparison to DEF(AUST) 5679*; Issue: 1.1, May 12, 1999.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

APPENDIX C HANDBOOK SUPPLEMENTAL INFORMATION

C.1 DoD Authority and Standards

The following paragraphs highlight the authority provided to the acquisition professional to establish system safety and software safety programs. These paragraphs are quoted or summarized from various DoD directives, instructions, policies, and military standards. These sections define the mandated requirement for all DoD systems acquisition and development programs to incorporate safety requirements and analysis into the design, development, testing, and support of software being used to perform or control critical system functions. DoD directs the authority and responsibility for establishing and managing an effective software safety program to the highest level of program authority.

C.1.1 Department of Defense Directive 5000.01

DoDD 5000.01, *The Defense Acquisition System* (May 12, 2003), Paragraph E1.23 establishes the requirement and need for addressing safety throughout the acquisition process.

- **E1.23 Safety** – Safety shall be addressed throughout the acquisition process. Safety considerations include human (includes human and system interfaces), toxic and hazardous materials and substances, production and manufacturing, testing, facilities, logistical support, weapons, and munitions and explosives. All systems containing energetics shall comply with insensitive munitions criteria.
- **E1.29 Total Systems Approach** – The PM shall be the single point of accountability for accomplishing program objectives for total lifecycle systems management, including sustainment. The PM shall apply human systems integration (HSI) to optimize total system performance (hardware, software, and human), operational effectiveness, suitability, survivability, safety, and affordability. PMs shall consider supportability, lifecycle costs, performance, and schedule in making program decisions. Planning for operation and support and the estimation of total ownership costs shall begin as early as possible. Supportability, a key component of performance, shall be considered throughout the system lifecycle.

C.1.2 Department of Defense Instruction 5000.02

DoDI 5000.02, *Operation of the Defense Acquisition System*, December 8, 2008, provides requirements for system safety and health.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- **Table 2-1. Statutory Requirements Applicable to Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs** – The following requirements are statutory for both MDAPs and MAIS acquisition programs—Programmatic Environment, Safety, and Occupational Health Evaluation (PESHE), including a National Environmental Policy Act (NEPA)/Executive Order (E.O.) 12114 Compliance Schedule.
- **Table 2-2. Statutory Requirements Applicable to Acquisition Category (ACAT) II and Below Acquisition Programs** – Programmatic Environment, Safety, and Health Evaluation (including NEPA/E.O. 12114 Compliance Schedule).
- **Enclosure 8, Human Systems Integration, Paragraph 2.a. Human Factors Engineering** – The PM shall take steps (e.g., contract deliverables and Government/contractor IPT teams) to ensure ergonomics, human factors engineering, and cognitive engineering are employed during systems engineering over the life of the program to provide for effective human-machine interfaces and to meet HSI requirements. Where practicable and cost effective, system designs shall minimize or eliminate system characteristics that require excessive cognitive, physical, or sensory skills; entail extensive training or workload-intensive tasks; result in mission-critical errors; or produce safety or health hazards.
- **Enclosure 8, Human Systems Integration, Paragraph 2.f. Safety and Occupational Health** – The PM shall ensure that appropriate HSI and environment, safety, and occupational health (ESOH) efforts are integrated across disciplines and into systems engineering to determine system design characteristics that can minimize the risks of acute or chronic illness, disability, or death or injury to operators and maintainers; and enhance job performance and productivity of the personnel who operate, maintain, or support the system.
- **Enclosure 12, Systems Engineering, Paragraph 6. Environment, Safety, and Occupational Health** – The PM shall integrate ESOH risk management into the overall systems engineering process for all developmental and sustaining engineering activities. As part of risk reduction, the PM shall eliminate ESOH hazards where possible, and manage ESOH risks where hazards cannot be eliminated. The PM shall use the methodology in MIL-STD-882D, *DoD Standard Practice for System Safety* (Reference (bz)). PMs shall report on the status of ESOH risks and acceptance decisions at technical reviews. Acquisition program reviews and fielding decisions shall address the status of all High and Serious risks and applicable ESOH technology requirements. Prior to exposing people, equipment, or the environment to known system-related ESOH hazards, the PM shall document that the associated risks have been accepted by the following acceptance authorities: the CAE for high risks, PEO-level for Serious risks, and the PM for Medium and Low risks. The user representative shall be part of this process throughout the lifecycle and shall provide formal concurrence prior to all Serious and High risk acceptance decisions.
- **Paragraph 6a. PESHE** – The PM for all programs, regardless of ACAT level, shall prepare a PESHE which incorporates the MIL-STD-882D process and includes the identification of ESOH responsibilities; the strategy for integrating ESOH considerations into the systems engineering process; identification of ESOH risks and

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

their status; a description of the method for tracking hazards throughout the lifecycle of the system; identification of hazardous materials, wastes, and pollutants (discharges, emissions, and noise) associated with the system and plans for their minimization and safe disposal; and a Compliance Schedule covering all system-related activities for the NEPA (Sections 4321-4347 of title 42 of United States Code. (Reference (ac)) and E.O. 12114 (Reference (ad)). The Acquisition Strategy shall incorporate a summary of the PESHE, including the NEPA/E.O. 12114 compliance schedule.

- **Paragraph 6b. NEPA/E.O. 12114** – The PM shall conduct and document NEPA/E.O. 12114 analyses for which the PM is the action proponent. The PM shall provide system-specific analyses and data to support other organizations' NEPA and E.O. 12114 analyses. The CAE (or for joint programs, the CAE of the Lead Executive Component) or designee is the approval authority for system-related NEPA and E.O. 12114 documentation.
- **Paragraph 6c. Mishap Investigation Support** – PMs will support system-related Class A and B mishap investigations by providing analyses of hazards that contributed to the mishap and recommendations for materiel risk mitigation measures, especially those that minimize human errors.

C.1.3 Defense Acquisition Guidebook

The *Defense Acquisition Guidebook* provides guidance on risk management that includes technical risks such as safety-significant risks. The Guidebook also provides guidance on implementing effective ESOH programs.

- **Environment, Safety, and Occupational Health –** As part of the program's overall cost, schedule, and performance risk reduction, the Program Manager shall prevent ESOH hazards, where possible, and manage ESOH hazards where they cannot be avoided (see 6.2.4.1, 6.2.5.2, and 6.2.5.3). More specifically, the Guidebook establishes requirements for Program Managers to manage ESOH risks for their system's lifecycle. The PM is required to have a PESHE document at Milestone B (or Program Initiation for ships) that describes
  - The strategy for integrating ESOH consideration into the systems engineering risk management process using the methodologies described in MIL-STD-882D
  - The schedule for completing NEPA and Executive Order 12114 documentation
  - The status of ESOH risk management, including a summary of the PESHE
  - From MS B on, the PESHE serves as a repository for top-level management information on ESOH risk
  - The identification, assessment, mitigation, residual risk acceptance, and ongoing evaluations of mitigation effectiveness and NEPA compliance.
- **Networks and Automated Information System (AIS) ESOH Management** – As noted in Table E3.T1 and Paragraph E7.1.6 of DoDI 5000.2, networks and automated

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

system programs, including those using COTS solutions, are not exempt from the DoD 5000 requirements to manage ESOH considerations as part of the systems engineering process and are required to document those efforts in a PESHE. The Automated Information System PM should perform the ESOH analyses appropriate for the scope of the acquisition program (e.g., software; hardware; and installation of facilities, fiber optic cables, and radio antennae). AIS programs that primarily deal with new or modified software applications should focus the PESHE on software system safety processes, procedures, and results. The PESHE for an AIS program that also involves hardware or facilities should also address ESOH considerations such as man-machine interface, identification of hazardous materials, preparation of required NEPA documentation, demilitarization planning, and disposal in accordance with hazardous waste laws and regulations.

Section 6.2.5 of the *Defense Acquisition Guidebook* provides further guidance on safety and occupational health topics.

C.1.4 Military Standards

C.1.4.1 MIL-STD-882B, Notice 1

MIL-STD-882B, *System Safety Program Requirements*, March 30, 1984 (Notice 1 – July 1, 1987), is used for numerous Government programs which were contracted during the 1980s prior to the issuance of MIL-STD-882C. The objective of this standard is the establishment of an SSP to ensure that safety, consistent with mission requirements, is designed into systems, subsystems, equipment, facilities, and interfaces. The authors of this standard recognized the safety risk that influences software presented in safety-critical systems. The standard provides guidance and specific tasks for the development team to address software, hardware, system, and human interfaces. These include the 300-series tasks. The purpose of each task is as follows:

- **Task 301, Software Requirements Analysis** – Task 301 requires the contractor to perform and document a Software Requirements Hazard Analysis. The contractor shall examine both system and software requirements, as well as the design, in order to identify unsafe modes for resolution, such as out-of-sequence, wrong event, inappropriate magnitude, inadvertent command, adverse environment, deadlocking, and failure-to-command. The analysis shall examine safety-critical computer software components at a gross level to obtain an initial safety evaluation of the software system.
- **Task 302, Top-Level Design Hazard Analysis** – Task 302 requires the contractor to perform and document a Top-Level Design Hazard Analysis. The contractor shall analyze the top-level design using the results of the Safety Requirements Hazard Analysis, if previously accomplished. This analysis shall include the definition and subsequent analysis of safety-critical computer software components, identification of the degree of risk involved, and the design and test plan to be implemented. The

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

analysis shall be substantially complete before the software-detailed design is started. The results of the analysis shall be present at the Preliminary Design Review.

- **Task 303, Detailed Design Hazard Analysis** – Task 303 requires the contractor to perform and document a Detailed Design Hazard Analysis. The contractor shall analyze the software detailed design using the results of the Software Requirements Hazard Analysis and the Top-Level Design Hazard Analysis to verify the incorporation of safety requirements and to analyze safety-critical computer software components. This analysis shall be substantially complete before coding of the software begins. The results of the analysis shall be presented at the Critical Design Review.
- **Task 304, Code-Level Software Hazard Analysis** – Task 304 requires the contractor to perform and document a Code-Level Software Hazard Analysis. Using the results of the Detailed Design Hazard Analysis, the contractor shall analyze program code and system interfaces for events, faults, and conditions that could cause or contribute to undesired events affecting safety. This analysis commences when coding begins and will continue throughout the system lifecycle.
- **Task 305, Software Safety Testing** – Task 305 requires the contractor to perform and document software safety testing to ensure that all hazards have been eliminated or controlled to an acceptable level of risk.
- **Task 306, Software User/Interface Analysis** – Task 306 requires the contractor to perform and document a Software/User Interface Analysis and the development of software user procedures.
- **Task 307, Software Change Hazard Analysis** – Task 307 requires the contractor to perform and document a Software Change Hazard Analysis. The contractor shall analyze all changes, modifications, and patches made to the software for safety hazards.

C.1.4.2 MIL-STD-882C

MIL-STD-882C, *System Safety Program Requirements* (January 19, 1993), establishes the requirement for detailed system safety engineering and management activities on all system procurements within the Department of Defense. This includes the integration of software safety within the context of the SSP. Although MIL-STD-882B and MIL-STD-882C remain on older contracts within DoD, MIL-STD-882D is the current system safety standard as of the date of this Handbook.

- **Paragraph 4, General Requirements and Paragraph 4.1, System Safety Program –** The contractor shall establish and maintain an SSP to support efficient and effective achievement of overall system safety objectives.
- **Paragraph 4.2, System Safety Objectives –** The SSP shall define a systematic approach to ensure that hazards associated with each system are identified, tracked, evaluated, and eliminated, or that the associated risk is reduced to a level acceptable to the PA throughout the lifecycle of a system.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- **Paragraph 4.3, System Safety Design Requirements –** System safety design requirements include designing software control and monitor functions to minimize hazardous events or mishaps.
- **Task 202, Preliminary Hazard Analysis, Section 202.2, Task Description –** The PHA shall take into account safety-related interface considerations among various elements of the system (e.g., material compatibilities, electromagnetic interference, inadvertent activation, fire and explosive initiation and propagation, and hardware and software controls at a minimum). This shall include consideration of the potential contribution by software (including software developed by other contractors or sources) to subsystem and system mishaps. Safety design criteria to control safety-critical software commands and responses (e.g., inadvertent command, failure to command, untimely command or responses, inappropriate magnitude, or PA-designated undesired events) shall be identified and appropriate actions taken to incorporate them into the software (and related hardware) specifications.

Task 202 is included as a representative description of tasks integrating software safety. The general description is also applicable to all other tasks specified in MIL-STD-882C. Software safety must be an integral part of system safety and software development.

C.1.4.3 MIL-STD-882D

MIL-STD 882D, *Standard Practice for System Safety*, replaced MIL-STD-882C in January 2000. Although the new standard is different from its predecessors, it continues to require system developers to document the approach to:

- Satisfy the requirements of the standard
- Identify hazards in the system through a systematic analysis approach
- Assess the severity of the hazards
- Identify mitigation techniques
- Reduce mishap risk to an acceptable level
- Verify and validate mishap risk reduction
- Report residual risk to the PM.

This process is identical to the process described in the preceding versions of the standard without specifying programmatic particulars. The process described in this Handbook meets the requirements and intent of MIL-STD-882D.

Succeeding paragraphs in this Handbook relate to Military Standards 882B and 882C to focus on specific tasks as part of the system safety analysis process. The tasks, while no longer part of MIL-STD-882D, provide valuable guidance for performing various aspects of the SSS.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

A PM should not universally accept a developer's proposal to make a no-cost change to replace earlier versions of the 882 series standard with MIL-STD-882D. This could have significant implications on the conduct of the safety program, preventing the PM and the safety team from obtaining the specific data required to evaluate the safety of the system and software.

C.1.4.4 DOD-STD-2167A

Although replaced by MIL-STD-498 in 1994, DOD-STD-2167A, *Defense Systems Software Development* (February 29, 1988), remains on numerous older contracts within DoD. This standard establishes uniform requirements for software development applicable throughout the system lifecycle. The requirements of this standard provide the basis for Government insight into a contractor's software development, testing, and evaluation efforts. The primary requirement of the standard establishes a system safety interface with the software development process.

- **Paragraph 4.2.3, Safety Analysis** – The contractor shall perform the analysis necessary to ensure that the software requirements, design, and operating procedures minimize the potential for hazardous conditions during the operational mission. Any potentially hazardous conditions or operating procedures shall be clearly defined and documented.

C.1.4.5 MIL-STD-498

IEEE and EIA Standard 12207 replaced MIL-STD-498 as the governing document for software development and documentation in military system procurements. MIL-STD-498,[20] *Software Development and Documentation* (December 5, 1994), Paragraph 4.2.4.1, established an interface with system safety engineering and defines the safety activities required for incorporation into software development throughout the acquisition lifecycle. This standard merged DOD-STD-2167A and DOD-STD-7935A to define a set of activities and documentation suitable for the development of both weapon systems and automated information systems. Other changes include improved compatibility with incremental and evolutionary development models; improved compatibility with non-hierarchical design methods; improved compatibility with Computer-Aided Software Engineering (CASE) tools; alternatives to, and more flexibility in, preparing documents; clearer requirements for incorporating reusable software; introduction of software management indicators; added emphasis on software support; and improved links to systems engineering. This standard can be applied in any phase of the system lifecycle.

- **Paragraph 4.2.4.1, Safety Assurance** – The developer shall identify as safety-critical those CSCI or portions thereof whose failure could lead to a hazardous system state (i.e., could result in death, injury, loss of property, or environmental harm). For such software, the developer shall develop a safety assurance strategy, including tests

20 IEEE 12207 and Joint Standard-016 replaced MIL-STD-498 in May 1998.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

and analyses, to ensure that the requirements, design, implementation, and operating procedures for the identified software minimize or eliminate the potential for hazardous conditions. The strategy shall include a software safety program that shall be integrated with the SSP, if one exists. The developer shall record the strategy in the SDP; implement the strategy; and produce evidence, as part of required software products, that the safety assurance strategy has been carried out.

In the case of reusable software products, including COTS, MIL-STD-498 stated that:

- **Appendix B, B.3, Evaluating Reusable Software Products** – General criteria shall be the software product's ability to meet specified requirements and be cost effective over the life of the system. Non-mandatory examples of specific criteria include, but are not limited to, the ability to provide required safety, security, and privacy.

C.1.5 Other Government Agencies

Other Governmental agencies are also interested in the development of safe software and are aggressively pursuing the development or adoption of new regulations, standards, and guidance for establishing and implementing software SSPs for their developing systems. Those Governmental agencies expressing an interest and actively participating in the development of this Handbook are identified below. The authoritative documentation used by these agencies to establish the requirement for a SwSSP is also included.

C.1.5.1 Department of Transportation

C.1.5.1.1 Federal Aviation Administration

FAA Order 1810, *Acquisition Policy*, establishes general policies and the framework for acquisition for all programs that require operational or support needs for the Federal Aviation Administration. The order implements the Department of Transportation Major Acquisition Policy and Procedures and consolidates the contents of more than 140 FAA orders, standards, and other references.

FAA Order 8040.4 requires all business lines to implement a formal risk management program consistent with their line of business. The order requires the use of a formal, disciplined, and documented decision-making process to address safety risks in relation to high-consequence decisions impacting the complete lifecycle.[21]

A significant FAA software development document is RTCA DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*. Important points from this resource include:

21 *FAA System Safety Handbook*; December 30, 2000.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- **Paragraph 1.1, Purpose** – The purpose of this document is to provide guidelines for the production of software for airborne systems and equipment that performs the intended function with a level of confidence in safety that complies with airworthiness requirements.
- **Paragraph 2.1.1, Information Flow from System Processes to Software Processes** – The system safety assessment (SSA) process determines and categorizes the failure conditions of the system. Within the system safety assessment process, an analysis of the system design defines safety-related requirements that specify the desired immunity from, and system responses to, these failure conditions. These requirements are defined for hardware and software to preclude or limit the effects of faults, and may provide fault detection and fault tolerance. As decisions are being made during the hardware design and software development processes, the system safety assessment process analyzes the resulting system design to verify that it satisfies safety-related requirements.

The software safety requirements are inputs to the software lifecycle process. To ensure that safety requirements are properly implemented, the system requirements typically include or reference:

- The system description and hardware definition
- Certification requirements, including Federal Aviation Regulations (United States), Joint Aviation Regulations (Europe), and Advisory Circulars (United States)
- System requirements allocated to software, including functional requirements, performance requirements, and software safety requirements
- Software level(s) and data substantiating determinations, failure conditions, Hazard Risk Index categories, and related functions allocated to software
- Software strategies and design constraints, including design methods such as partitioning, dissimilarity, redundancy, and safety monitoring
- The software safety requirements and failure conditions if the system is a component of another system.

System lifecycle processes may specify requirements for software lifecycle processes to aid system verification activities.

C.1.5.1.2 Aerospace Recommended Practice

The Society of Automotive Engineers provides two standards representing Aerospace Recommended Practice (ARP) to guide the development of complex aircraft systems. ARP4754 presents guidelines for the development of highly integrated or complex aircraft systems with particular emphasis on electronic systems. While safety is a key concern, the ARP covers the complete development process. ARP4761 is a companion standard that contains detailed

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

guidance and examples of safety assessment procedures. These standards could be applied across application domains, but some aspects are avionics specific.[22]

The avionics risk assessment framework is based on Development Assurance Levels, which are similar to DEF(AUST) 5679, *Safety Integrity Levels*. The process assigns a DAL to each functional failure condition identified under ARP4754 and ARP4761, based on the severity of the effects of the failure condition identified in the Functional Hazard Assessment. However, the severity corresponds to levels of aircraft controllability rather than direct levels of harm. The result does not consider the likelihood of accident sequences in the initial risk assessment.

The DAL of an item may be reduced if the system architecture:

- Provides multiple implementations of a function (redundancy)
- Isolates potential faults in part of the system (partitioning)
- Provides for active (automated) monitoring of the item
- Provides for human recognition or mitigation of failure conditions.

The ARPs provide detailed guidance on these issues. The preliminary SSA provides the justification for the reduction.

DALs are provided with equivalent numerical failure rates so that quantitative assessments of risk can be made. However, the effectiveness of particular design strategies cannot always be quantified, and qualitative judgments are often required. In particular, the ARPs make no attempt to interpret the assurance levels of software in probabilistic terms. Like DEF(AUST) 5679, the software assurance levels are used to determine the techniques and measures to be applied in the development processes.

When the development is sufficiently mature, actual failure rates of hardware components are estimated and combined by the SSA to provide an estimate of the functional failure rates. The assessment should determine if the corresponding DAL has been met. To achieve the objectives, the SSA suggests using a failure modes and effects analysis and a fault tree analysis, which are described in the appendices of ARP4761, *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.*[23]

22 *International Standards Survey and Comparison to DEF(AUST) 5679*; Document ID: CA38809-101; Issue: 1.1; May 12, 1999; page 3.
23 Ibid. pages 27 and 28.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

C.1.5.2 Department of Homeland Security

C.1.5.2.1 Coast Guard

Commandant Instruction M5000.10, *Major Systems Acquisition Manual* (November 29, 2006), establishes policy, procedures, and guidance for the administration of Coast Guard major acquisition projects. The manual addresses system safety and the associated hazard analyses as part of logistics support elements during design and development.[24]

Using MIL-STD-498 as a foundation, the Coast Guard developed Commandant Instruction M5234-4, *Software Development and Documentation Standards* (January 21, 2003), for internal Coast Guard use. The document establishes the Coast Guard's software development and documentation standards and requirements and assigns oversight responsibility to the Commandant (G-S). The standard specifically addresses safety requirements for software. Paragraph 4.2.2.1 requires the evaluation of reusable software for safety. Paragraph 4.2.3 requires developers to establish a software safety program that is fully integrated with the system safety program and to perform safety analyses of the developmental software. The standard also requires developers to document the hazard analyses and list any software safety requirements identified during the analyses.

C.1.5.3 National Aeronautics and Space Administration

NASA has been developing safety-critical, software-intensive aeronautical and space systems for many years. To support the required planning of software safety activities on research and operational procurements, NASA published NASA Software Safety Standard 8719.13A in September 1997. This standard provides a methodology for software safety in NASA programs and describes the activities necessary to ensure that safety is designed into software that is acquired or developed by NASA. Several DoD and Military Standards, including MIL-STD-882, *System Safety Program Requirements*, influenced the development of this NASA standard.

The standard requires that the software safety process:

- Ensure that the system and subsystem safety analyses identify safety-critical software; any software that has the potential to cause a hazard or is required to support control of a hazard, as identified by safety analyses, is safety-critical software
- Ensure that the system and subsystem safety analyses clearly identify the key inputs into the software requirements specification (e.g., identification of hazardous commands, limits, interrelationship of limits, sequence of events, timing constraints, voting logic, and failure tolerance)
- Ensure that the development of the software requirements specification includes the software safety requirements that have been identified by software safety analyses

24 Commandant Instruction M5000.10, Appendix A, Section 3.10.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Ensure that the software design and implementation incorporate the software safety requirements
- Ensure that the appropriate V&V requirements are established to ensure proper implementation of the software safety requirements. This includes an assessment of the scope and level of IV&V to be planned and implemented based on the level of criticality and risk of the software application. A statement will be made in either the program/project plan or the software development plan as to the level of IV&V to be accomplished.
- Ensure that test plans and procedures will satisfy the intent of the software safety verification requirements
- Ensure that the results of the software safety verification effort are satisfactory.

C.1.5.4 Food and Drug Administration

The U.S. Food and Drug Administration (FDA) is responsible for ensuring the safety of a wide variety of products, from food to food processing equipment to medicines and medical devices to cosmetics. With the rapid increase in the use of software in systems, the FDA has developed requirements and guidelines for the use of software in systems for assessing the safety of systems developed for the FDA. The Division of Electrical and Software Engineering of the FDA's Office of Science and Engineering Laboratory has a software laboratory tasked with developing analytical methods and tools for ensuring the safety and security of software used for medical purposes.

C.1.6 Commercial

Unlike the historical relationship established between DoD agencies and contractors, commercial companies are not obligated to a specified, quantifiable level of safety risk management for the products they produce (unless contractually obligated through a subcontract arrangement with another company or agency). Instead, the primary motivation in the commercial sector is economical, ethical, and legal liability factors. For those commercial companies that are motivated or compelled to pursue the elimination of, or control the, safety risk in software, several commercial standards are available for guidance. This Handbook references a few of the more commonly used standards. While these commercial standards are readily accessible, few provide the practitioner with a defined software safety process or the "how-to" guidance required to implement the process. However, the processes described in this Handbook are generally compatible with those required by the commercial standards.

C.1.6.1 Institute of Electrical and Electronic Engineering

C.1.6.1.1 IEEE STD 1228-1994

The Institute of Electrical and Electronic Engineers published IEEE STD 1228-1994, *IEEE Standard for Software Safety Plans*, to describe the minimum acceptable requirements for the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

content of a software safety plan. This standard contains four clauses. Clause 1 discusses the application of the standard. Clause 2 lists references to other standards. Clause 3 provides a set of definitions and acronyms used in the standard. Clause 4 contains the required content of a software safety plan. An informative annex is included and discusses software safety analyses. IEEE STD 1228-1994 is voluntary and is written for individuals responsible for defining, planning, implementing, or supporting software safety plans. This standard closely follows the methodology of MIL-STD-882B, Change Notice 1.

C.1.6.1.2 IEEE/EIA STD 12207

IEEE/ EIA Standard 12207 replaced MIL-STD-498 as the governing document for software development and documentation in military system procurements. Originally developed by the International Electrotechnical Commission and the International Organization for Standardization, IEEE/EIA STD12207 is an international standard for software development. The IEEE/EIA standard implements the IEC/ISO standard for industries in the United States. The standard addresses the safety aspects of the software during all phases of development, from the procurement planning stages through lifecycle maintenance. The standard requires that the developer prepare a plan that describes the safety processes with respect to software. The developer must describe their plan to identify and document software safety requirements, including an assessment of the criticality of these requirements. The standard also requires the developer to maintain configuration control over safety-significant software to ensure that modifications do not inadvertently affect the safety of the final product. Finally, the standard requires the verification of software safety requirements and design implementation using suitably rigorous methods.

The IEEE/EIA 12207 standard provides little guidance on the identification of safety-significant functions in the software. Therefore, users must either develop their own methodology or rely on other sources, such as this Handbook, for guidance.

C.1.6.2 TechAmerica G-48 System Safety Committee

The G-48 System Safety Committee is an advisory body made up of system safety experts from TechAmerica member companies. TechAmerica was formed in 2009 from a merger of the Information Technology Association of America (ITAA) and the AeA (formerly the American Electronics Association, which had merged earlier with the Government Electronics and Information Technology Association (GEIA). In October 2008, ITAA published GEIA-STD-0010, *Standard Best Practices for System Safety Program Development and Execution*. On February 12, 2009, after the formation of TechAmerica, the standard was approved by the American National Standards Institute and was re-designated ANSI/GEIA-STD-0010-2009.

ANSI/GEIA-STD-0010-2009 includes discussion of, and guidance for, current best practices in software safety. The standard's Appendix A, "Guidance for Implementation of a System Safety Effort," includes sections on assessing software criticality and performing an FHA, the detailed steps of which are spelled out in Task 208 of the standard. Section A.6 of the standard, titled

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

"Software System Safety Engineering Analysis and Integrity," covers many aspects of software safety and emphasizes that a successful software safety engineering activity is based on both a hazard analysis process and a software integrity process.

C.1.6.3 International Electrotechnical Commission

The IEC developed IEC-61508 in December 1997. The standard addresses safety systems incorporating Electrical/Electronic/Programmable Electronic Systems (E/E/PES). IEC-61508 also provides an applicable framework for safety systems regardless of the technology on which those systems are based (e.g., mechanical, hydraulic, or pneumatic). This standard provides a generic approach for all safety lifecycle activities for systems comprised of E/E/PES that are used to perform safety functions.[25]

IEC-61508 consists of seven parts:

- Part 1: General Requirements
- Part 2: Requirements for E/E/PES
- Part 3: Software Requirements
- Part 4: Definitions
- Part 5: Guidelines on the Application of Part 1
- Part 6: Guidelines on the Application of Part 2 and Part 3
- Part 7: Bibliography of Techniques.

The standard addresses all relevant safety lifecycle phases when an E/E/PES performs safety functions. The standard was created with rapidly developing technology in mind. The framework in this standard is considered to be sufficiently robust and comprehensive to cater to future developments. Although IEC 61508 is directed toward systems that perform safety functions, the methodology and processes described are also applicable to systems that perform safety-significant functions.

C.2 International Standards

C.2.1 Australian Defense Standard 5679

DEF(AUST) 5679, published by the Australian Department of Defense in March 1999, is a standard for the procurement of safety-critical systems with an emphasis on computer based systems. The standard focuses on safety management and the phased production of safety assurance throughout the system development lifecycle, with emphasis on software and

25 IEC 61508-1, Edition. 1; December 1997; page 6.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

software-like processes. A safety case provides auditable evidence of the safety assurance argument.

Software risk and integrity assessments are based on the concept of development integrity levels. Probabilistic interpretations of risk are excluded because of the scope for error or corruption in the quantitative analysis process, and because it is "currently impossible to interpret or assess low targets of failure rates for software or complex designs."[26]

For each potential accident identified by the PHA, a severity category (Catastrophic, Fatal, Severe, or Minor) is allocated, based on the level of injury incurred. Sequences of events that could lead to each accident are identified and are assigned a probability where estimation is possible.

The Defense Standard uses Level of Trust (LOT) definitions. One of seven LOTs is allocated to each system safety requirement, depending on the severity category of the accidents that may result from the corresponding system hazard. The LOT may be reduced if each accident sequence can be shown to be sufficiently improbable. Each LOT defines the desired level of confidence that the corresponding system safety requirement will be met.

Next, one of seven SILs is assigned to each Component Safety Requirement (CSR), indicating the level of rigor required for meeting the CSR. By default, the SIL level of the CSR is the same as the LOT of the system safety requirement corresponding to the CSR. However, the default SIL may be reduced by up to two levels by implementing fault-tolerant measures in the design to reduce the likelihood of the corresponding hazard.

Appendices to the standard provide guidance on the identification of safety-significant functions, software safety requirements, guidance on establishing the safety program, and other useful guidelines. This standard requires the use of formal notation and mathematical proofs of correctness for software identified as safety critical.

C.2.2 United Kingdom Defence Standard 00-56

United Kingdom Defence Standard (DEF-STAN) 00-56, *Safety Management Requirements for Defence Systems*, Issue 4, June, 2007, supersedes and combines DEF-STANs 00-56, 00-54, and 00-55. The standard provides requirements and guidelines for the development of all defense systems, not solely computer-based systems. Like MIL-STD-882D, DEF-STAN 00-56 provides broad guidance on the requirements for a system safety program to achieve a level of safety risk that is as low as reasonably practicable. The DEF-STAN has a strong basis in UK law and notes that the system developer, as the duty holder, has the responsibility to exercise due diligence in the development of the system.

---

26 *International Standards Survey and Comparison to DEF(AUST) 5679*; Document ID: CA38809-101; Issue: 1.1; May 12, 1999; page 3.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

A key aspect of the DEF-STAN is the development of a Safety Case. The DEF-STAN requires the developer to provide a Safety Case at various stages of development. Early Safety Cases document the planned approach to risk evaluation and mitigation; latter Safety Cases provide the evidence of risk mitigation. The Safety Case should demonstrate how safety will be, is being, and has been achieved and maintained. The Safety Case should consist of a structured argument supported by a body of evidence. The quantity and quality of the evidence depends on the systems risks, complexity, and unfamiliarity of the circumstances involved. The unfamiliarity criteria are aimed at novel systems or unusual environments. The standard requires the developer to work closely with the Ministry of Defense and stakeholders throughout the development process to ensure that the delivered system achieves the stated requirements.

The aspects of software are dealt with in part 2 of the standard. The concept of complex electronic elements, which includes software and custom hardware (e.g. firmware is treated the same as software), is used.

C.3 Proposed Contents of the System Safety Data Library

C.3.1 System Safety Program Plan

The SSPP is a requirement of MIL-STD-882 for Department of Defense procurements. The SSPP details tasks and activities of the system safety engineering and management program established by the supplier. The SSPP also describes the engineering processes to be employed to identify, document, evaluate, and eliminate or control system hazards to the levels of acceptable risk for the program. The approved plan (by the customer) provides the formal basis of understanding and agreement between the supplier and the customer on how the program will be executed to meet contractual requirements. Specific provisions of the SSPP include program scope and objectives, system safety organization and program interfaces, program milestones, general safety requirements and provisions, and specific hazard analyses to be performed. Details must be provided for the methods and processes to be employed on the program to identify hazards and failure modes, derive design requirement to eliminate or control the hazard, and the test requirements and verifications methods to be used to ensure that hazards are controlled to acceptable levels. Even if the contract does not require an SSPP, the safety manager of the development agency should produce this document to reduce interface and safety process misconceptions as it applies to the design and test groups. Templates for the SSPP format are located in DID, DI-SAFT-80100, and MIL-STD-882C.

Specifically, the plan must include:

- A general description of the program
- The system safety organization
- System safety program milestones
- System safety program requirements

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Hazard analyses to be performed
- Requirements analysis to be performed
- Functional analysis to be performed
- Hazard analysis processes to be implemented
- Hazard analyses data to be obtained
- Method of safety verification
- Training requirements
- Applicable audit methods
- Mishap prevention, reporting, and investigation methods
- System safety interfaces
- PESHE requirements.

If the SSPP is not accomplished, numerous problems can surface. Most importantly, a "roadmap" of hazard identification, mitigation, elimination, and risk reduction effort is not presented for the program. The result is the reduction of programmatic and technical support and resource allocation to the SSP. The SSPP is not simply a map of task descriptions of what the safety engineer is to accomplish on a program, rather it is an integration of safety engineering across critical management and technical interfaces. The SSPP allows all integrated program team members to assess and agree to the necessary support required by all technical disciplines of the development team to support the safety program. Without this support, the safety design effort will likely fail.

C.3.2 Software Safety Program Plan

Some contractual deliverable obligations will include the development of a Software Safety Program Plan. This was a common requirement in the mid-1980s through the early 1990s. The original intent was to formally document that a development program (that was software intensive or that had software performing safety-critical functions) considered the processes, tasks, interfaces, and methods to ensure software was taken into consideration from a safety perspective. In addition, the intent was to ensure that steps were taken in the design, code, test, and IV&V activities to minimize, eliminate, or control the safety risk to a predetermined level. At the same time, many within the safety and software communities mistakenly considered software safety to be a completely separate engineering task rather than the original system safety engineering activities.

Today, it is recognized that software must be rendered safe (to the greatest extent possible equating to the lowest safety risk) through a systems methodology. Safety experts agree that the ramifications of software performing safety-critical functions can be assessed and controlled through sound system safety engineering and software engineering techniques. In fact, without the identification of specific software-caused or software-influenced system hazards or failure modes, the assessment of residual safety risk of software cannot be accomplished. Although software error identification, error trapping, fault tolerance, and error removal are essential in

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

software development, without a direct tie to a system hazard or failure mode, the product is usually not safety significant—it is reliability and system availability.

If possible, and in accordance with MIL-STD-882, specific tasks, processes, and activities associated with the elimination or minimization of software-specific safety risk should be integrated into the SSPP. If a customer is convinced that a separate program plan is specifically required for software safety, this can be accomplished. The plan should contain a description of the safety, systems, and software engineering processes to be employed to identify, document, evaluate, and eliminate or control system hazards (that are software caused or software influenced) to the levels of acceptable risk for the program. The plan should describe program interfaces and lines of communication between technical and programmatic functions and should document suspense and milestone activities according to an approved schedule. As a reminder, the software safety schedule of events should correspond with the software (and hardware) development lifecycle and be in concert with each development and test plan published by other technical disciplines associated with the program. IEEE STD 1228-1994, *IEEE Standard for Software Safety Plans*, provides an industry standard for the preparation and content of a SwSPP. An outline based on this standard is presented in Figure C-1 and can be used as a guide for plan development.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Figure C-1: Contents of a SwSPP - IEEE STD 1228-1994

C.3.3 Preliminary Hazard List

The purpose of a PHL is to compile of a list of preliminary hazards of the system as early in the development lifecycle as possible. Sources of information that assist the analyst in compiling a preliminary list are:

- Similar systems hazard analysis
- Historical mishap data
- Lessons learned
- Trade study results
- Functional analysis
- Preliminary requirements and specifications
- Design requirements from design handbooks

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Potential hazards identified by safety team brainstorming
- Generic software safety requirements and guidelines
- Common sense.

The list of preliminary hazards of the proposed system becomes the basis of the PHA and the consideration and development of design alternatives. The PHL must be used as inputs to proposed design alternatives and trade studies. As the design matures, the list is reviewed to eliminate those hazards that are not applicable to the proposed system and to document and categorize those hazards deemed to contain inherent (potential) safety risk.

Lessons learned information can be extracted from databases established specifically to document lessons learned in design, manufacture, fabrication, test, and operation activities or from actual mishap information from organizations such as the DoD Service safety agencies. Each hazard that is identified should be recorded on the list and contain the source of reference. Each hazard identified should fulfill the "Source – Mechanism – Outcome" criteria where source represents the failure modes, mechanism represents the hazard title, and outcome represents the mishap.

C.3.4 Safety-Critical Functions List

Although not currently identified in MIL-STD-882, the introduction of an SCFL historically became important as specific process steps to perform software safety tasks became more mature and defined. The design, code, test, IV&V, implementation, and support of software code can become expensive and resource limited. Software code that performs safety-critical functions requires an extensive protocol or level of rigor within design, code, test, and support activities. This added structure, discipline, and level of effort add cost to the program and should be performed first on those modules of code that are most critical from a safety perspective. Conversely, code developed with no identified inherent safety risk would require a lesser level of design, test, and verification protocol. Documenting the safety-critical functions early in the concept exploration phase identifies those software functions or modules that are safety critical by definition.

The point of the SCFL is to ensure that software code or modules of code that perform safety-critical functions are defined and prioritized as safety-critical code or modules. This is based on credible, system-level functions that have been identified as safety critical. The safety-critical identifier on software establishes the design, code, test, and IV&V activities that must be accomplished to ensure the software is safety risk minimized.

The identification of the SCFL is a multi-discipline activity initiated by the safety manager. Identification of the preliminary safety functions of the system requires inputs from systems, design, safety, reliability engineering, project managers, and the user. It requires early assessment of the system concepts, specifications, requirements, and lessons learned. Assessing the system design concepts includes analysis of:

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Operational functions
- Maintenance and support functions
- Test activities
- Transportation and handling
- Operator and maintainer personnel safety
- Software/hardware interfaces
- Software/human interfaces
- Environmental health and safety
- Explosive constraints
- Product loss prevention.

Identified safety-critical functions should be documented, tracked, and matured as the design matures. A distinct safety-critical function in preliminary design may be completely eliminated in detailed design, or a function could be added to the preliminary list as the design matures or customer requirements change.

C.3.5 Preliminary Hazard Analysis

The PHA activity is a safety engineering and software safety engineering function performed to identify the hazards and preliminary causal factors of the system under development. The hazards are formally documented to include information regarding the description of the hazard, causal factors, the effects of the hazard, and preliminary design requirements for hazard control by mitigating each cause. Performing the analysis includes assessing hazardous components, safety-significant interfaces between subsystems, environmental constraints, operation, test and support activities, emergency procedures, test and support facilities, and safety-significant equipment and safeguards. The PHA format is defined in DI-SAFT-80101A of MIL-STD-882C. This DID also defines the format and contents of all hazard analysis reports. An example of a PHA hazard record format is provided in Figure 4-26.

The PHA also provides an initial assessment of mishap severity and probability of occurrence. At this point, the probability assessment is usually subjective and qualitative.

To support the tasks and activities of a software safety effort, the causes of the root hazard must be assessed and analyzed. These causes should be separated into four separate categories:

- Hardware initiated causes
- Software initiated causes
- Human error initiated causes
- Human error causes that were influenced by software input to the user/operator.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

This categorization of causes assists in the separation and derivation of specific design requirements that are attributed to software. Both software-initiated causes and human error causes influenced by software input must be adequately communicated to the systems engineers and software engineers for identification of software design requirements to preclude the initiation of the root hazard identified in the analysis.

The PHA becomes the input document and information for all other hazard analyses performed on the system. This includes the SSHA, SHA, Health Hazard Assessment (HHA), and O&SHA.

C.3.6 Subsystem Hazard Analysis

The hazard analysis performed on individual subsystems of the (total) system is the Subsystem Hazard Analysis. This analysis is launched from the individual hazard records of the PHA which were identified as a logically distinct portion of a subsystem. Although the PHA is the starting point of the SSHA, it must be only that— a starting point. The SSHA is a more in-depth analysis of the functional relationships between components and equipment (this also includes the software) of the subsystem. Areas of consideration in the analysis include performance, performance degradation, functional failures, timing errors, design errors, or inadvertent functioning. Failure detection, failure isolation, failure annunciation, and control entity corrective action are also considered at the subsystem level.

As previously stated, the SSHA is a more in-depth analysis than the PHA. This analysis begins to provide the evidence of requirement implementation by matching hazard causal factors to elements of the design to prove or disprove hazard mitigation. The information that must be recorded in the SSHA includes, but is not limited to, hazard descriptions, all hazard causes (hardware, software, human error, or software-influenced human error), hazard effects, and derived requirements to either eliminate or reduce the risk of the hazard by mitigating each causal factor. The inverse of a hazard cause can usually result in a derived requirement. The analysis should also define preliminary requirements for safety warning or control systems, protective equipment, and procedures and training. Also of importance in the data record is the documentation of the design phase of the program, component(s) affected, component identification per drawing number, initial hazard RAC (which includes probability and severity prior to implementation of design requirements), and the record status (opened, closed, monitored, deferred, etc.).

From a software safety perspective, the SSHA must define those hazards or failure modes that are specifically caused by erroneous, incomplete, or missing specifications (including control software algorithm elements, interface inputs and outputs, and threshold numbers), software inputs, or human error (influenced by software furnished information). These records are the basis for the derivation and identification of software requirements that eliminate or minimize the safety risk associated with the hazard. The SSHA must initiate resolution of how the system or subsystem will react if the software error does occur. Fault resolution scenarios must consider the reaction of the subsystem or system if a potential software error (failure mode) becomes a reality. For example, if a potential error occurs, does the system power down, detect the error

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

and correct it, go to a lesser operational state, fail soft, fail safe, fail operational, fail catastrophic, or some combination of these?

C.3.7 System Hazard Analysis

The SHA provides documentary evidence of safety analyses of the subsystem interfaces and system functional, physical, and zonal requirements. As the SSHA identifies the specific and unique hazards of the subsystem, the SHA identifies those hazards introduced to the system by the interfaces between subsystems, man/machine interfaces, and hardware/software interfaces. The SHA assesses the entire system as a unit and evaluates the mishaps, hazards, failure modes, and causal factors that could be introduced through system physical and functional integration. An example of the minimum information to be documented in an SSHA and SHA is provided in Figure C-2.

Figure C-2: SSHA and SHA Hazard Record Example

Although interface identification criteria are not required or defined in the SSHA, they are the primary source of preliminary data to assist in a "first cut" of the SHA. The SHA is accomplished later in the design lifecycle (after the PDR and before the CDR), which increases the cost of design requirements that may be introduced as an output of this analysis. Introducing new requirements late in the development process also reduces the possibility of completely eliminating the hazard through the implementation of design requirements. It is recommended

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

that initial interfaces be considered as early as possible in the PHA and SSHA phases of the safety analysis. Having this data in preliminary form allows the maturation of the analysis in the SHA phase of the program to be timelier.

C.3.8 Safety Requirements Verification Report

A common misconception of system safety engineering is that the most important products produced by the analysis are the hazard analysis reports (PHA, SSHA, SHA, HHA, and O&SHA). This is only partially correct. The primary product of system safety engineering analysis is the identification and communication of requirements to eliminate or reduce the safety risk associated with the design, manufacture, fabrication, test, operation, and support of the system. Once identified, these requirements must be verified to be necessary, complete, correct, and testable for the system design. Not only is it the responsibility of the system safety function to identify, document, track, and trace hazard mitigation requirements to the design, but also to identify, document, participate in, or perform the verification activities.

These activities can vary according to the depth of importance communicated by program management, design and systems engineering, and system safety. If a program is hindered by limited resources, this verification may be as simple as having the design engineers review the hazard records under their responsibility, verify that all entries are correct, and verify that the designers have incorporated all the derived requirements associated with the record. On the other hand, the verification activities may be as complicated as initializing and analyzing specific testing activities and analyzing test results; the physical checking of as-built drawings and schematics; the physical checking of components and subsystems during manufacture; and the physical review of technical orders, procedures, and training documentation.

The Safety Requirements Verification Report provides the audit trail for the formal closure of hazard records at System Safety Group (SSG) meetings. This report is not a mandatory requirement if the hazard tracking database contains the necessary fields documenting the evidence of safety requirement implementation (Figure C-3). Before closing a hazard record, the PM must be assured that requirements functionally derived or identified via the hazard record have been incorporated and implemented. There must also be assurance that generic software safety requirements have been adequately implemented. Those requirements not implemented must be identified in the assessment of residual risk and integrated in the final RAC of the hazard record. This formal documentation is required information that must be presented to the test and user organizations. These organizations must be convinced that all identified hazards and failure modes have been minimized to acceptable levels of safety risk prior to system-level testing.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

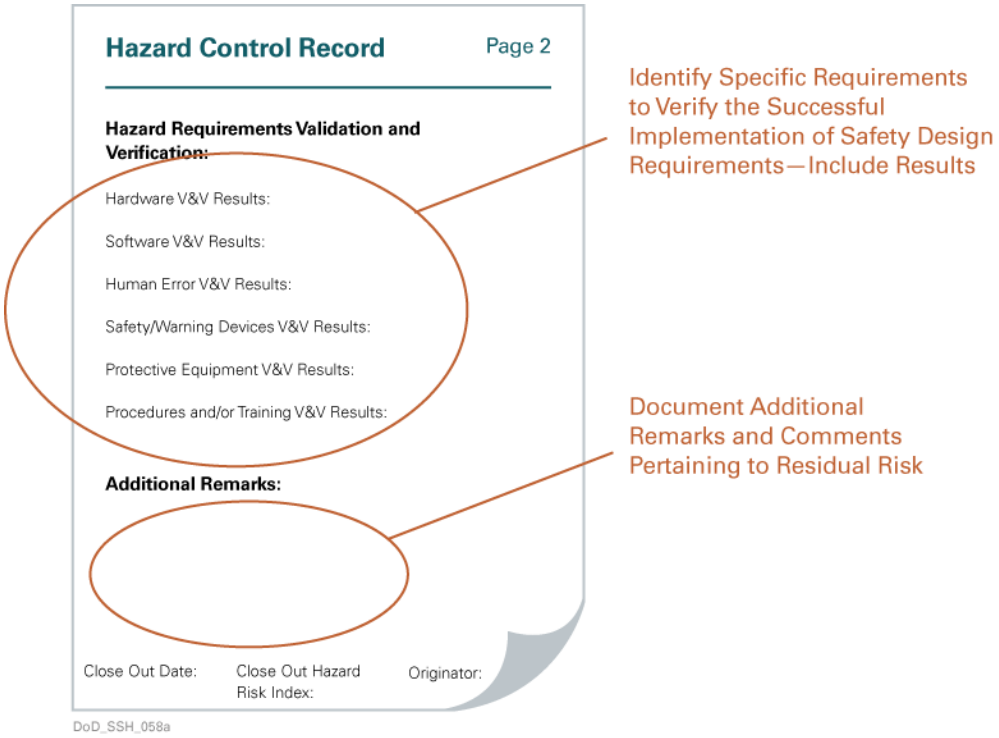

Figure C-3: Hazard Requirements Verification Document Example

C.4 Contractual Documentation

C.4.1 Statement of Operational Need

A product lifecycle begins with the Statement of Need (SON), a product of the Mission Area Analysis (MAA). The MAA identifies deficiencies and shortfalls in defense capabilities or defines more effective ways of accomplishing existing tasks. The purpose of the SON is "to describe each need in operational terms relating to planned operations and support concepts" [Air Force Regulation 57-1]. This document will provide the software safety team with a definition of operational need of the product and an appreciation of where the concept was originated (including assumptions).

Most SONs do not have specific or specified statements regarding design, manufacture, fabrication, test, operation and support, or software safety. If they did, the planning, support, and funding for system safety engineering would be easier to secure in the initial phases of systems development. Since the SON will most likely lack any safety-significant statement, it should be reviewed for the purpose of understanding the background, needs, desires, requirements, and specifications of the ultimate system user. This helps in the identification of scope, breadth, and depth of the software safety program and facilitates an understanding of what the user considers

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

to be acceptable in terms of safety risk. With little (or no) verbiage regarding safety in the SON, a communication link with the user (and customer, if different from the user) is essential.

C.4.2 Request for Proposal

Although not always specifically addressed, many modern-day RFPs include an implied requirement for a system safety engineering program which may include software safety activities. In today's environment of software-intensive systems, an implied requirement is no longer acceptable. The user must request a specified SSP, applicable safety criteria, definitions of acceptable risk, levels of required support, and anticipated deliverables required by the customer.

The primary reason for detailed safety criteria in the RFP is the establishment and documentation of the defined scope and the level of effort for a contract proposal and subsequent SSP. This establishes a design definition that can be accurately planned, budgeted, and implemented. One of the biggest obstacles for most SSPs is the lack of sufficient budget and program resources to adequately accomplish the system safety task in sufficient detail. This is often due to insufficient criteria detail in the RFP, SOW, and contract. Without this detail, the developer may incorrectly bid the scope of the system safety and software safety portion of the contract. The level of detail in the RFP is the responsibility of the user.

For the developer, planning at this stage of the program consists of a dialog with the customer to identify specifically those requirements or activities that the user perceives to be essential for reducing the safety risk of software performing safety-critical functions. Safety managers, in conjunction with the software development manager, assess the specific software safety requirements to fully understand the customer desires, needs, and requirements. This should be predicated on detailed safety criteria documented in the RFP. Dialog and communication are required by the developer and the customer to ensure that each safety program requirement is specifically addressed in the proposal. A sample RFP statement for safety is included in Appendix G.

C.4.3 Contract

Specific contract implications regarding system safety and software safety engineering and management are predicated on proposal language, contract type, and specific contract deliverables. The software safety team must ensure that all outside (the software safety team) influences are assessed and analyzed for impact to the breadth and depth of the program. For example, a development contract may be Cost-Plus, Award-Fee. Specific performance criteria of the safety and software team may be tied directly to the contract award fee formula. If this is the case, the software safety team will be influenced by contractual language. Additionally, the specifics of the contract deliverables regarding system safety criteria for the program will also influence the details defined in the SSPP. The software safety team must establish the program to meet the contract and contractual deliverable requirements.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

C.4.4 Statement of Work

The program SOW and its parent contract are mandatory reading for the software safety engineer and the software safety team members associated with the program. Planning and scoping the software safety program, processes and tasks, and products to be produced are all predicated on the contract and SOW. The contract and SOW define contractual requirements, scope of activity, and required deliverables. The contract and SOW will become the launch pad for the development of the SSPP and either the software safety appendix or the SwSPP. The SSPP defines how the developer will meet all program objectives, accomplish/produce the required contract deliverables, and meet scheduled milestone activities. When approved, the SSPP is normally a contractually binding document.

In many cases, developing and coordinating the system safety paragraph contents of an SOW is one of the most unplanned and uncoordinated activities that exists between the customer and the supplier. On numerous occasions, the customer does not know specifically what they want, so the SOW is intentionally vague, or conversely calls for all tasks of a regulatory standard (i.e., MIL-STD-882). In other instances, the SOW is left vague as to "not stifle contractor innovation." Most system safety activities required by the SOW are not coordinated and agreed upon by the contractor or the customer. This problem could be minimized if sufficient details were provided in the RFP. Program Managers and technical managers must realize that specific criteria for a program's breadth and depth can be adequately stated in an RFP and SOW without dictating to the developer how to accomplish the specified activities.

Unfortunately, the RFP and SOW seldom establish the baseline criteria for an adequate SSP. With this in mind, two facts must be considered. First, the SOW is usually an approved and contractually binding document that defines the breadth of the safety work to be performed. Second, it is usually the depth of the work to be performed that has the potential for disagreement. This is often predicated on the lack of specified criteria in contractual documents and lack of allocated critical resources by program management to support these activities. Planning for system safety activities that will be contractually required should be an activity initiated by the PA. It is the responsibility of the customer to understand the resource, expertise, and schedule limitations and constraints of the contractor performing the safety analysis. This knowledge can be useful in scoping SOW requirements to ensure that a practical program is accomplished which identifies, documents, tracks, and resolves the hazards associated with the design, development, manufacture, test, and operation of a system. A well-coordinated SOW safety requirement should accurately define the depth of the SSP and define the necessary contract deliverables. The depth of the system safety and software safety program can be scoped in the contractual language if the customer adequately defines the criteria for the following:

- The goals and objectives of the system safety design, test, and implementation effort
- Technical and educational requirements of the system safety manager and engineers
- The allocated system loss rate requirements to be allocated for the subsystem design efforts

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- The specific category definitions pertaining to mishap probability and severity for the RAC and the SCM
- The specific software control category definitions
- The required LOR tasks for each identified software criticality level
- The defined system safety engineering process and methods to be incorporated
- The specific scope of effort and closeout criteria required for each category of RAC hazards
- The required contractual safety deliverables, including acceptable format requirements.

For DoD procurements, current versions of both system safety and software development standards should be used in the preparation of the SOW software safety tasks, as shown in Figure C-4. While these military standards may not be available in future years, their commercial counterpart should be available for use as professional organizations keep them current.

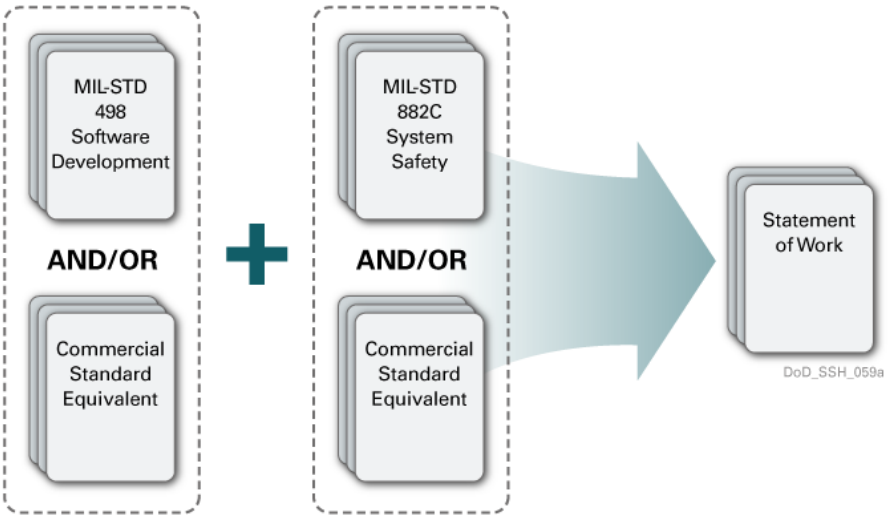

Figure C-4: Software Safety SOW Paragraphs

Although not recommended, if the contractor is tasked with providing a draft SOW as part of the proposal activity, the customer (safety manager) should carefully review the proposed paragraphs pertaining to system safety and resolve potential conflicts prior to the approval of the document. Examples of SOW/SOO system safety paragraphs are provided in Appendix G.

C.4.5 System and Product Specification

The system specification identifies the technical and mission performance requirements of the product to be produced. The system specification allocates requirements to functional areas,

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

documents design constraints, and defines the interfaces between or among the functional areas. The system specification must be thoroughly reviewed by the safety manager and software safety team members to ensure those physical and functional specifications are completely understood. This will assist in the identification and scope (breadth and depth) of the analysis to be performed. It also provides an understanding of the technologies involved, the magnitude of the managerial and technical effort, assumptions, limitations, engineering challenges, and the inherent safety risk of the program.

The system specification should identify and document any quantified safety-critical requirements that must be addressed. For example, an aircraft development program may have a system specification for vehicle loss rate of $1x10^{-6}$. It is the responsibility of the safety manager, in concert with design engineering, to allocate this loss rate to each effected subsystem. Each major subsystem would be allocated a portion of the loss rate requirement such that any design activity that causes a negative impact to the loss rate allocation would be flagged and resolved. The safety and engineering leads are required to track how close each design activity is to their vehicle loss rate allocation. If each design team meets or exceeds their allocation, the aircraft will meet its design specification. A side note to this example, Air Force accident records indicate that the vehicle loss rate of operational aircraft is an order of magnitude less than the vehicle loss rate from which the aircraft was originally designed.

There is information in the system specification that will influence the methods, techniques, and activities of the system safety and software safety engineering teams. The SSPP and/or SwSPP must reflect specific activities to meet the defined requirements of system and user specifications.

C.4.6 System and Subsystem Requirements

Another essential set of documents that must be reviewed by the system safety manager and engineers is the initial requirements documentation. In some instances, these initial requirements are delivered with the RFP, or they can be derived as a joint activity between the developer and the customer. Knowledge and understanding of the initial system and subsystem requirements allow the safety engineers to begin activities associated with the PHL and PHA. This understanding allows for greater fidelity in the initial analysis and reduces the time required in assessing design concepts that may not be in the design at PDR.

For preliminary software design activities, the generic software safety requirements and guidelines must be part of the initial requirements. An example of these preliminary (generic) requirements and guidelines is provided in Appendix E. This list or similar lists must be thoroughly reviewed and analyzed, and only those that are deemed appropriate will be provided to the software design team.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

C.5 Planning Interfaces

C.5.1 Engineering Management

The lead engineer or engineering manager is pivotal in ensuring that each subsystem, system, interface, and support engineer provides the required support to the system safety team. This support must be timely to support the analyses, contractual deliverables, and schedule milestones of the contract and the SOW. The engineering support must also be timely to effectively influence the design, development, and test of the system, each subsystem, and their associated interfaces. The engineering manager has direct control over the allocation of engineering resources (including the engineers themselves) for the direct support of system safety engineering analyses. Without the support of the lead engineer, analyses performed by the safety team would have to be based on "best guess" assumptions and inaccurate engineering data and information. Engineering management:

- Coordinates the activities of the supporting technical disciplines
- Manages technical resource allocation
- Provides technical input to the Program Manager
- Defines, reviews, and comments on the technical adequacy of safety analyses
- Ensures that safety is an integral part of system engineering and is allocated sufficient resources to conduct analyses.

C.5.2 Design Engineering

The hardware and software design engineers must rely on the completeness and correctness of requirements derived by support functions such as safety, reliability, maintainability, and quality. The derivation of these requirements cannot be provided from a vacuum (e.g., a lone analyst producing requirements without understanding the functional and physical interfaces of the system). Design engineers are also dependent on the validity and fidelity of the system safety process to produce credible requirements. Safety-specific requirements are obtained from two sources—generic safety requirements and guidelines (see Appendix F) and derived requirements produced through hazard analysis and failure mode analysis. Generic safety requirements are normally derivatives of lessons learned and requirements identified on similar systems. Regardless of the source of the safety requirement, it is essential that the design engineer understands the intent of each requirement and the ramifications of not implementing that requirement in the design.

Correctness, completeness, and testability are also mandatory attributes of safety requirements. In most instances, the correctness and completeness of safety requirements are predicated on the communication of credible hazards and failure modes to the designer of a particular subsystem. Once the designer understands the hazard, specific requirements to eliminate or control the hazard are derived in concert by the safety engineer and design engineer.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The ultimate product of a system safety engineering activity is the elimination or control of the risk associated with hazards and failure modes. The incorporation and implementation of safety-specific design requirements accomplish this task. Design engineering must have intimate knowledge of the system safety and software safety activities and the intent of the safety requirements derived. The knowledge of the safety processes and the intent of the requirements by the design team establish credibility for the safety engineer to actively perform within the design function.

C.5.3 Systems Engineering

Systems Engineering is defined as "an interdisciplinary approach to evolve and verify an integrated and optimally balanced set of product and process designs that satisfy user needs and provide information for management decision making." [MIL-STD-499B Draft]

"Systems engineering, by definition, involves design and management of a total system comprised of both hardware and software. Hardware and software are given equal weight in analysis, tradeoffs, and engineering methodology. In the past, the software portion was viewed as a subsidiary, follow-on activity. The new focus in systems engineering is to treat both software and hardware concurrently in an integrated manner. At the point in the system design where the hardware and software components are addressed separately, modern engineering concepts and practices must be employed for software, the same as hardware." [STSC 1, 1994]

The overall objectives of systems engineering, [DSMC 1990] are to perform the following:

- Ensure that system definition and design reflects requirements for all system elements, including equipment, software, personnel, facilities, and data
- Integrate technical efforts of the design team specialists to produce an optimally-balanced design
- Provide a comprehensive indentured framework of system requirements for use as performance, design, interface, support, production, and test criteria
- Provide source data for development of technical plans and contract work statements
- Provide a system framework for logistic analysis, Integrated Logistic Support (ILS) trade studies, and logistic documentation
- Provide a system framework for production engineering analysis, producibility trade studies, and production manufacturing documentation
- Ensure that lifecycle cost consideration and requirements are fully considered in all phases of the design process.

Each of the preceding objectives is important to system safety engineering and has the potential to impact the overall safety of the system in development. Ultimately, system engineering has the responsibility for developing and incorporating all design requirements that meet the system operational specifications. This includes the safety-specific requirements for hardware, software,

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

and human interfaces. Systems engineering must be assured that system safety has identified and implemented a sound approach for identifying and resolving system hazards and failure modes. A systems engineering approach supports system safety engineering and the MIL-STD-882 safety precedence to design for minimum risk.

Systems engineering must be in agreement with the processes and methods of the safety engineer to ensure safety requirements are incorporated in the design of the system. This process must have the ability to efficiently identify, document, track, trace, test, and validate requirements which reduce the safety risk of the system.

C.5.4 Software Development

The software engineering/development interface is one of the most critical interfaces for a successful software safety engineering activity. The software engineering team must understand the need associated with the analysis and review tasks of system safety engineering. They must also comprehend the methods to be used and the utility and products of the methods in the fulfillment of the software safety tasks. This interface is new to most software developers. Although they usually understand the necessity of controlling the safety risk of software performing in safety-critical systems, their view of system safety engineering principles is somewhat limited. It the responsibility of the SSS team to assess the software development acquisition process and determine when, where, and how to be most effective in tracing software safety requirements to test. The software engineering team must support the SSS team by allocating specific personnel to the team to assist in the safety activities. This usually consists of personnel that can address software design, code, test, IV&V, CM, and quality control functions.

C.5.5 Integrated Logistics Support

An inherent difference between hardware support and software support is that hardware support is based on the finished product, while software support must mimic the development process. Hardware support must use the tools necessary to repair a finished product, not tools required to build one. Software support, on the other hand, must use tools functionally identical to those used during the development process.

Lifecycle support strategies typically span the support spectrum from sole source contractor to full Government organic, with each strategy presenting different advantages and disadvantages. A high level IPT consisting of the operational user, the PEO, and the acquisition agent must make the support decision prior to Milestone A. This focuses attention on the software support process and allows the acquisition agent to begin planning earlier in the program.

The Computer Resources Integrated Support Document (CRISD) is the key document for software support. The CRISD determines facility requirements; specifies equipment and required support software; and lists personnel number, skills, and required training. The CRISD contains information crucial to the establishment of the software engineering environment (SEE),

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

its functionality, and limitations. The CRISD is a management tool that accurately characterizes the SEE's evolution over time. [STSC, 1996]

From a software safety perspective, the software support environment must be aware of the safety implications of the software to be maintained and supported in the operational environment. Safety-critical functions and their relationships to controlling software must be fully defined. Any software code that directs, functions, or controls safety-critical functions of the system must be fully defined and communicated in the software support documentation. Any software ECP or maintenance support function pertaining to safety-critical modules of code must be thoroughly reviewed by the SSS team prior to implementation.

C.5.6 Other Engineering Support

Each program development process will have unique differences and specific interface requirements. For example, a potential customer may require that all system hazard probabilities be quantified to a specific confidence level and that no qualitative engineering estimates will be acceptable. This requirement forces the safety engineering analysis to be heavily predicated on the outputs of reliability engineering. Although interfaces with reliability engineering are common for most DoD procurements, the example demonstrates the necessity of cultivating programmatic and technical interfaces based on contractual, technical, and programmatic obligations. Other technical interface examples may include human factors, quality assurance, reliability engineering, supportability, maintainability, survivability, test and evaluation, IV&V, and ILS.

Planning for and securing agreement with the managerial and technical interface precludes the necessity of trying to wedge it into defined processes at a later date. Each program is unique and presents planning challenges and peculiar interfaces.

C.6 Meetings and Reviews

C.6.1 Program Management Reviews

Program Management Reviews (PMRs) are normally set up on a routine schedule to allow the Program Manager to obtain an update of the program status. The update will consist of the review of significant events since the previous PMR, short-term and long-term schedules, financial reports and budgeting forecasts, and technical contributions and limitations in design. Timely and valid inputs to the PMR provide the Program Manager and other key program personnel with the information necessary to make informed decisions that affect the program. This may include the allocation of critical resources.

System safety may or may not have briefing items on the PMR agenda, depending on whether the PM/director specifically requires the presentation of the current status of the safety program, and whether safety issues should be raised to this decision-making level. Safety issues should

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

only be raised to this level of program management if it is essential to resolve an issue of a critical nature (e.g., resolution of an important safety issue could not be attained at any lower level of technical or managerial support in the time required to solve the problem). It is recommended that the safety manager attend the PMRs for programmatic visibility and to keep abreast of program milestones, limitations, and technical contributions.

C.6.2 Integrated Product Team Meetings

The IPT is the team of individuals that ensures integrated product development (IPD). IPD is a philosophy that systematically employs a teaming of functional disciplines to integrate and concurrently apply all necessary processes to produce an effective and efficient product that satisfies the customer's needs.[27] IPD applies to the entire lifecycle of a product.

The IPT provides a technical-managerial framework for a multi-disciplinary team to define the product. The IPT emphasizes up-front requirements definition, trade-off studies, and the establishment of a change control process for use throughout the entire lifecycle. From a system safety perspective, IPTs effect how the processes defined in a TEMP are integrated for the development and support required by safety throughout the project. This includes special considerations, methods, and techniques defined by IPT members and supporting technical disciplines.

C.6.3 System Requirements Reviews

System Requirements Reviews are normally conducted during the concept exploration or demonstration/validation phase of a program to ensure that system-level functional analysis is relatively mature and that system-level requirements have been allocated. The purpose is to ensure that system requirements have been completely and correctly identified and that mutual agreement is reached between the developer and the customer. Particular emphasis is placed on ensuring that adequate consideration has been given to logistic support, safety, software, test, and production constraints.

Primary documents used in this review consist of the documentation products of the system requirement allocation process. This includes functional analysis, trade studies, functional flow block diagrams, requirement allocation sheets, and the requirements allocation reports from other disciplines. This also encompasses such disciplines as reliability, supportability, maintainability, human factors, and system safety.

The system safety manager must ensure that the safety requirements have been thoroughly captured and that they cover known hazards. The list of known (or suspected) hazards from the PHL or PHA should be used as a guide to check against each system and subsystem requirement.

---

27 Air Force Material Command Manual, *Acquisition Management Acquisition Logistics Management*, January 19, 1995 (DRAFT).

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

For safety verification of new requirements, the system and subsystem requirements must then be individually checked to ensure that new hazards have not been introduced. Requirements should be traceable to systems, subsystems, and their respective interfaces (e.g., human/machine, hardware/software, and system/environment), as well as to specific system and subsystems hazards and failure modes. Traceability of allocated requirements to the capability of the system to meet the mission needs and program objectives within planned resource constraints must be demonstrated by correlation of technical and cost information. [DSMC 1990]

Requirements considered safety critical must already be defined and documented. A formal method to flag these requirements in documentation must be in place, and any attempt to change or modify these requirements must be limited. The documentation process must include a checks and balances approach that notifies the safety manager if safety-critical functions or requirements are considered for change, modification, or elimination.

C.6.4 System and Subsystem Design Reviews

The System Design Review is one of the final activities of the demonstration/validation phase of the program, and thus becomes the initial review of the engineering/manufacturing development phase. The purpose of the SDR is to assess and evaluate the optimization, traceability, correlation, completeness, and risk of the system-level design that fulfills the system functional baseline requirements. The review assesses and evaluates total system requirements for hardware, software, facilities, personnel, and preliminary logistic support. This review also assesses the systems engineering activities that help establish the products that define the system. These products include trade studies, functional analysis and allocation, risk analysis, mission requirements, manufacturing methods and processes, system effectiveness, integrated test planning, and configuration control.

C.6.5 Preliminary Design Review

Preliminary Design Reviews are normally conducted for each hardware and software configuration item (or functionally grouped CIs) after top-level design efforts are complete and prior to the start of detailed design. The PDR is usually held after the approval of the development specifications and prior to system-level reviews. The review focuses on technical risk, preliminary design (including drawings) of system elements, and traceability of technical requirements. Specific documentation for CI reviews includes development specifications, trade studies supporting preliminary design, layout drawings, engineering analysis (including safety hazard analysis, human engineering, failure modes and effects analysis, and ILS), interface requirements, mock-ups and prototypes, computer software top-level design, and software test plans. Special attention is given to interface documentation, high-risk areas, long lead-time procurement, and system-level trade studies.

The safety engineer must provide:

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- The GSSRs and guidelines allocated to the system functional baseline
- The initial SRA and identified CSSRs of the preliminary design architecture
- The FHA, SCFL, and LOR requirements of the software assurance and integrity program
- Results of any safety-significant trade studies
- The MSSRs derived through the FHA, PHL, and PHA
- The specific MSSRs derived through the initial draft efforts of the SSHA, SHA, and O&SHA.

The primary focus of the safety team is to ensure that the generic and the specific derived requirements are documented in the requirements specification and communicated to the appropriate design function, and that the safety intent of each is adequately displayed in the proposed design. The primary output of the PDR is the assurance that safety-specific requirements (and their intent) are being adequately implemented in the design. The assurance provided is a two-way traceability of requirements to specific hazards and failure modes and to the design itself.

C.6.6 Critical Design Review

The Critical Design Review is conducted for each hardware and software CI before release of the design for manufacturing, fabrication, and configuration control. For software, the CDR is conducted prior to coding and preliminary software testing. The CDR discloses the detailed design of each CI as a draft product specification and related engineering drawings. The design becomes the basis for final production planning and initial fabrication. In the case of software, the completion of the CDR initiates the development of source and object code. Specific review items for a CDR include detailed engineering drawings, interface control drawings, prototype hardware, manufacturing plans, and Quality Assurance Plans (QAPs).

The safety engineer must provide:

- All safety specific requirements (CSSRs, GSSRs, and MSSRs) derived from the safety analyses activities. This includes trade studies; functional and physical analyses; and FHA, SSHA, SHA, O&SHA, and HHA activities.
- Evidence that all SRA safety requirements and guidelines are included in the design architecture for hardware, software, and human interfaces
- The finalized SCFL and evidence of LOR requirements implementation
- All safety-specific testing requirements to verify the implementation of safety requirements in the design
- The initial safety risk assessment of the system or SoS.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The CDR provides evidence to the safety engineer that the design meets the goals and objectives of the system safety program and that the designers have implemented each safety requirement in the final design. An approved CDR normally places the design under formal configuration control.

The safety team must leave the CDR with:

- Assurance of requirements traceability from analysis to design
- Evidence that the final design meets the goals and objectives of the SSPP
- Evidence that safety design requirements are verifiable, including requirements that must be verified through testing
- Evidence (quantifiable, if practical) of residual safety risk in the design.

C.6.7 Test Readiness Review

The Test Readiness Review is a formal review of the developer's readiness to proceed with CI testing. For software development, it is the readiness to begin CSCI-level testing of that specific CSCI. The TRR ensures that the safety requirements have been properly built into the system and subsystem, and that safety test procedures are complete and mature. The TRR also ensures that residual safety risk is defined and understood by program management and engineering. The TRR verifies that all system safety functionality and requirements have been incorporated through verification methods.

Another safety aspect of the TRR is the safety implications of the test itself. There are usually greater safety implications on the test of hardware systems compared to the testing of software CSCIs and CSUs. Safety inputs to the TRR include:

- Safety analysis of the test itself
- GO/NO-GO test criteria
- Emergency test shut-down procedures
- Emergency response procedures
- Test success or failure criteria.

If the specific test is for the verification of safety requirements, safety inputs to the TRR must also include:

- Documentation of the safety requirements to be verified during the test activities
- Credible "normal" and "abnormal" parameters of test inputs
- Expected response to normal and abnormal test parameters and inputs

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Methodology to document test outputs and results for data reduction and analysis
- Methodology to determine test success or failure.

To support the TRR for embedded software programs, the safety manager must verify the traceability of safety requirements between system hazards and failure modes and specific modules of code. Failure to identify the hazard-to-requirement will impact the ability to ensure that a hazard has been adequately mitigated until the safety test is run or test results are reviewed. This could result in software that does not contain adequate safety control for further testing or final delivery. Further possible ramifications include a requirement to re-engineer the code (influencing cost and schedule risks) or resulting in unknown residual safety risk.

C.6.8 Functional Configuration Audit

The Functional Configuration Audit (FCA), while from a legacy standard, still has a place in the spectrum in that all functions expected to be assessed for safety need to be compiled into the SAR. The objective of the FCA is to verify that CI performance complies with the hardware and software development interface requirement specifications. Hardware and software performance is verified by test data in accordance with its functional and allocated configuration. FCAs on complex CIs may be performed on an incremental basis; however, FCAs must be performed prior to release of the configuration to a production activity.

The safety team must provide all functional safety requirements and functional interface requirements recommended for the configuration audit. These requirements must be prioritized for safety and should include those that influence safety-critical functions. The safety team must ensure that the configuration verifies the implementation of the safety design requirements and that those requiring verification have been verified. Verification of safety requirements must be traceable throughout the system's functional configuration and in the hazard record.

For software, agreement is reached on the validity and completeness of the Software Test Reports. The FCA process provides technical assurance that the software safely performs the functions against respective CSCI requirements and operational and support documentation.

C.6.9 Physical Configuration Audit

While the Physical Configuration Audit (PCA) is a legacy audit, the SAR must still address the expected configuration of all hardware, software, and firmware that comprises the SoS or system from a safety analysis and test viewpoint. The PCA is the formal examination of the "as-built" version of the configuration item against the specification documentation that established the product baseline. Upon approval of the PCA activity, the CI is placed under formal CM control.

The PCA establishes test and QA acceptance requirements and criteria for production units. The PCA process ensures that a detailed audit is performed on documentation associated with

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

engineering drawings, specifications, technical data, and test results. On complex CIs, the PCA may be accomplished in three phases—review of the production baseline, operational audit, and customer acceptance of the product baseline.

As with a PCA of hardware, the PCA for software is a formal technical examination of the as-built product against its design. The design may include attributes that may not be customer requirements. If this situation exists, these attributes must be assessed from a system safety perspective. The requirements from the SSHA and other software safety analyses (including physical interface requirements) will be compared with the closure of software-significant hazards as a result of design. Test results will be assessed to ensure that requirements are verified. In addition, the implemented design will be compared to as-built code documentation to verify that it has not been altered after testing (except for configuration control changes).

From a safety perspective, the most effective manner to conduct the audit is to target critical safety requirements. It is recommended that the PCA auditors choose Catastrophic and Critical hazards to verify "as-built" safety of the particular CI. The design, control, and test of safety requirements often involve the most complex and fault tolerant code and architectures. As a consequence, they are often performed late in the testing schedule, giving a clear picture of the CI status.

C.7 Working Groups

C.7.1 System Safety Working Group

The SSWG consists of individuals with expertise to discuss, develop, and present solutions for unresolved safety issues to program management or design engineering. The SSWG investigates engineering problem areas assigned by the SSG and proposes alternative design solutions to minimize safety risk. The requirement to have an SSWG is predicated on the need to resolve programmatic or technical issues. The SSWG charter should describe the intent and workings of the SSWG and the relationships to program management, design engineering, support functions, test agencies, and field activities.

The SSWG assists the safety manager in achieving the system safety and software safety objectives of the SSMP. To prepare for an SSWG, the safety manager for the development organization must develop a working group agenda from an itemized list of issues to be discussed. Programmatic and technical support required for the meeting should also be considered. The safety manager must ensure that the appropriate individuals are invited to the working group meeting and have had ample time to prepare to meet the objectives of the agenda.

The results of the SSWG are formally documented in the meeting minutes. The minutes should include a list of attendees, the final agenda, copies of support documentation (including presentations), documented resolutions, agreements, recommendations to program management and engineering, and allocated action items.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

C.7.2 Software System Safety Working Group

The Software System Safety Working Group is chaired by the software safety point of contact and may be co-chaired by the PA's PFS or System Safety Program Manager. The SwSSWG is often the primary means of communication between the PA and the developer's software safety program. To be effective, the roles and responsibilities of the SwSSWG members, the overall authority and responsibilities, and the procedures and processes for operation of the SwSSWG must be clearly defined. Each SwSSWG must have a charter. This charter is generally appended to the SSPP. The charter describes:

- The authority and responsibilities of the safety and software safety POCs
- The roles and responsibilities of the membership
- The process and tasks to be implemented to accomplish the software safety program
- The management activities to ensure the successful implementation of LOR tasks and requirements
- The processes to be used by the SwSSWG for accepting Hazard Action Reports and entering them into the hazard log
- The processes and procedures for risk resolution and acceptance
- The expected meeting schedule and frequency
- The process for closure of hazards for which the SwSSWG has closure authority.

The SwSSWG schedule includes periodic meetings, meetings prior to major program milestones, and splinter meetings as required to fully address the software safety issues. Membership includes:

- The safety principals (e.g., safety point of contact, the PA PFS, and safety program manager(s))
- System safety engineering
- Software safety engineering
- Systems engineering
- Software engineering
- Software testing
- Independent verification and validation
- Software QA
- Software configuration management
- Human factors engineering.

To be most effective, the SwSSWG must include representatives from the user community. A core group of the SwSSWG, including the safety POCs, the PA PFS, system engineering, software engineering, and the user community, should be designated voting members of the IPT

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

if a voting structure is used for decision making. Ideally, however, all decisions should be by consensus.

The developing agency's software safety POC chairs or co-chairs (with the procuring agency's PFS for software safety) all formal SwSSWG meetings. The chair, co-chair, or designated secretariat prepares an agenda for each meeting. The agenda ensures that all members, including technical advisors, have time to address topics of concern. Any member of the SwSSWG IPT may request the addition of agenda items. The chair or secretariat forwards the agenda to all IPT members at least a week in advance of the meeting. Members wishing to include additional agenda items may contact the secretariat with the request prior to the meeting, such that the agenda presented at the opening of the meeting includes their topics.

The chair must maintain control of the meeting to ensure that side discussions and topics not germane to the software safety program do not prevent accomplishment of useful work. However, the group should work as a team and not be unnecessarily constrained.

The DA must make provisions for preparing and distributing meeting minutes. The meeting minutes formally record the results of the meeting, including any decisions made and the rationale behind the decisions. These decisions may include the selection of a hazard resolution or decisions regarding hazard closure and the engineering rationale used to arrive at that decision. The chair or secretariat distributes the meeting minutes to all IPT members and retains copies in the SDL.

C.7.3 Test Integration Working Group/Test Planning Working Group

The Test Integration Working Group and the Test Planning Working Group may be combined on smaller programs. These groups exist to ensure that planning is accomplished on the program to adequately provide test activities for developmental test, requirements verification test, and operational test. Developmental testing verifies that the design concepts, configurations, and requirements meet user and system specifications. Operational and support testing verifies that the systems and related components can be operated and maintained with the support concept of the user. This includes user personnel with the skill levels defined in the user specification to operate and support the system. Operational testing verifies that the system operates as intended.

C.7.4 Computer Resources Working Group

The Computer Resources Working Group (CRWG) is formed as early as possible during the concept exploration phase, but no later than Milestone 1.[28] The CRWG provides advice to the Program Manager regarding the software support concept; computer resources policy; plans; procedures; standards; and the TEMP, IV&V, and other development risk controls, all of which

28 *Mission Critical Computer Resources Management Guide*, Defense Systems Management College, Fort Belvoir, VA.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

are influenced by the level of safety risk. The CRWG also contributes to the program's configuration control with external groups and interfaces. The CRWG includes the implementing agency, using agency, support agencies, and any other DoD agency.

From a system safety perspective, participation on the CRWG is essential to provide and ensure that software safety processes are included in all policies, procedures, and processes of the software development team. This ensures that safety-specific software requirements are integrated into the software design, code, and test activities. The CRWG must recognize the importance of safety input into the design and test functions for developing safety-critical software.

C.7.5 Interface Control Working Group

The Interface Control Working Group (ICWG) is one of the program office's controls over external interfaces that affect the system under development.[29] The ICWG has purview over areas external to the product, such as interoperability and operational issues. The ICWG works in coordination with the CRWG. The ICWG coordinates current and proposed software and hardware interfaces.

The developer's ICWG has initial responsibility for the software and hardware interfaces of the system under development. The ICWG transfers this responsibility to the acquirer as the time for delivery of the software approaches. The ICWG coordinates and signs off on the interface requirements between developers and associate developers. All changes are reviewed by the ICWG before submittal to the acquirer's ICWG.

Software safety identifications and annotations to each change must accompany the initial interface coordination and any subsequent change to an interface to preserve safety and reduce residual safety risk.

C.8 Resource Allocation

C.8.1 Safety Personnel

Once the specific tasks are identified in accordance with the contract and the SOW, the safety manager must estimate the number of person-months that will be required to perform the safety tasks. This estimate will be based on the breadth and depth of the analysis to be performed, the number of contractual deliverables, and the funds obligated to perform the tasks. If the estimate of person-hours exceeds the allocated budget, the prioritized tasks that will be eliminated must be communicated to program management to ensure that they are in agreement with the decisions made, and that these discussion and agreements are fully documented. Conversely, if program management cannot agree with the level of effort to be performed (according to the

29 ibid., pg. 10-5

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

budget), they must commit to the allocation of supplemental resources to obtain the level of effort necessary to achieve the contractually required safety levels. Again, specific safety tasks must be prioritized and scoped (breadth and depth) to provide program management (and the customer) with enough information to make informed decisions regarding the level of safety effort for the program.

The personnel identified to accomplish the system safety and safety-significant software engineering tasks must be competent in the discipline. The DA should not be allowed by the PA to use "out of work" engineers to accomplish safety-significant tasks. Desired minimum qualifications of personnel assigned to perform the software safety portion of the system safety task include:

- Undergraduate degree in engineering or technically-related subject (e.g., chemistry, physics, mathematics, engineering technology, industrial technology, or computer science)
- System safety management course
- System safety analysis course
- Software safety engineering course
- Software acquisition and development course
- Systems engineering course
- Two to five years experience in system safety engineering or management.

C.8.2 Funding

Funds obligated to the program system safety effort must be sufficient to identify, document, and trace system safety requirements to eliminate or control hazards and failure modes to an acceptable level of safety risk. Section C.8.3 communicates the benefits of prioritizing tasks and program requirements for the purpose of allocating personnel to perform the safety tasks in accordance with the contract and the SOW. Sufficient funds must be allocated for the performance of system safety engineering which meets user and test requirements and specifications. Funding limitations and shortfalls must be communicated to the Program Manager for resolution. The PM can only make informed decisions if system safety processes, tasks, and deliverables are documented and prioritized in detail. This will help facilitate the allocation of funds to the program and identify the managerial, economical, technical, and safety risk of underfunding the program. In addition, the cost-benefit and technical ramifications of decisions can be formally documented to provide a detailed audit trail for the customer.

C.8.3 Safety Schedules and Milestones

The planning and management of a successful SSP is supplemented by the safety engineering and management program schedule. The schedule should include near-term and long-term events, milestones, and contractual deliverables. The schedule should also reflect the system

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

safety management and engineering tasks required for each lifecycle phase of the program and to support DoD milestone decisions. Also of importance is safety data required to support special safety boards that may be required for compliance and certification purposes. Examples include FAA certification, the Department of Defense Explosives Safety Board, nuclear certification, and the Non-Nuclear Munitions Safety Board. Each event, deliverable, and milestone should be tracked to ensure suspense and safety analysis activities are timely to facilitate cost-effective design solutions to meet the desired safety specifications of the system development activity and the customer.

Planning for the SSP must include the allocation of resources to support travel of safety management and engineers. The contractual obligations of the SOW, in concert with the processes stated in the program plans and the required support of program meetings, dictate the scope of safety involvement. With limited funds and resources, the system safety manager must determine and prioritize the level of support that will be allocated to program meetings and reviews. The number of meetings that require support, the number of safety personnel that are scheduled to attend, and the physical location of the meetings must be assessed against the budgeted travel allocations for the safety function. This activity (resource allocation) becomes complicated if meeting support priorities are not established up front. Once priorities are established, meetings that cannot be supported due to budget constraints can be communicated to program management for concurrence or the reallocation of resources with program management direction.

C.8.4 Safety Tools and Training

Planning the system safety engineering and management activities of a program should include the specific tools and training required for the accomplishment of the program. Individual training requirements should be identified for each member of the safety team that is specifically required for program-specific tasks and for the personal and professional growth of the analyst. Once the required training is identified, planning must be accomplished to secure funding and to inject training into the program schedule where most appropriate for impact minimization.

Tools that may be required to support the system safety activities are, in many instances, program specific. Examples include a fault tree analysis program to identify single-point failures and to quantify probability of occurrence, or a sneak-circuit analysis tool for sneak-circuit analysis. As wide and varied as individual or programmatic needs are, a safety database must be budgeted. A safety database is required to provide the audit trail necessary for the identification, documentation, tracking, and resolution of hazards and failure modes in the design, development, test, and operation of the system. This database should be flexible enough to document the derived requirements of the hazards analysis; create specific reports for design engineering, program management, and the customer; and document the traceability and verification methodology for the requirements implemented in the design of the system. Traceability includes the functional, physical, and logical links between the requirements; the hazards they were derived from; testing requirements to verify the requirements; and the specific modules of code that are affected. In addition, the safety database must be able to store the necessary

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

information normally included in Hazard Action Records specified by MIL-STD-882 and their associated DIDs. Examples of these reports include the PHA, FHA, SSHA, SHA, O&SHA, and Safety Review Board results.

Supplemental tools that may be required for the performance of software safety tasks include software timing analysis tools, CASE tools, functional or data flow analysis tools, and requirements traceability tools. The safety manager must discuss the requirements for software- and safety-significant tools with the software development manager to determine availability within the program.

C.8.5 Required Hardware and Software

Specific hardware and software resources should be identified when planning for the accomplishment of the SSP. Specific tools may be program dependent, including emulators, models and simulations, requirement verification tools, and timing/state analysis tools. The expenditures for required hardware and software must be identified and planned for up front (as possible). Hardware includes the necessary computer hardware (e.g., CPU, monitor, hard drives, and printers) required to support specific software requirements (e.g., fault tree analysis software, hazard tracking database, etc.). Development activities can be unique and may require program-specific hardware or software. This planning activity ensures that the safety manager considers all program objective hardware and software requirements.

C.9 Program Plans

System safety engineering and management activities must be thoroughly addressed as they interface with other managerial or technical disciplines. Although not specifically authored by the system safety manager or the software safety team, numerous program plans must have a system safety input for completeness. This safety input ensures that formal lines of communication, individual responsibilities and accountabilities, specific safety tasks, and process interfaces are defined, documented, and agreed upon by all affected functional or domain disciplines. Having each technical and functional discipline performing to approved and coordinated plans increases the probability of successfully meeting the goals and objectives of the plan.

Examples of specific program plans that require system safety input include (but are not limited to) the RMP, QAP, Reliability Engineering Plan (REP), SDP, SEMP, and TEMP. The system safety manager must assess the program management activities to identify other plans that may require interface requirements and inputs. Complete and detailed descriptions are located in systems design and engineering standards and textbooks. Individual program or system development documentation should contain the best descriptions that apply specifically to the system being developed.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

C.9.1 Risk Management Plan

The RMP describes the programmatic aspects of risk planning, identification, assessment, reduction, and management to be performed by the developer. The RMP should relate the developer approach for handling risk compared to the available options. Tailoring of the plan should reflect those program areas that have the greatest potential impact. This may be programmatic, technical (including safety), economical, or political risk to the program design, development, test, or operations activities. The plan should describe how an iterative risk assessment process is applied at all WBS levels for each identified risk as the design progresses and matures. The RMP should also describe how risk assessment is used in the technical design review process, configuration control process, and performance monitoring activities. In most cases and for most programs, safety risk is a subset of technical risk. The risk assessment and risk management activities of the program must include safety inputs on critical issues for informed decision making by both program management and engineering. Safety-critical and safety-related issues that are not coming to expected or acceptable closure through the defined safety resolution process must be communicated to the risk management group through systems engineering.

In the area of software safety, the following areas of risk must be considered within the RMP:

- Costs associated with the performance of specific software safety analysis tasks, including the costs associated with obtaining the necessary training to perform the tasks
- Schedule impact associated with the identification, implementation, test, and verification of safety-critical software requirements
- Technical risk associated with the failure to identify safety-critical design requirements early in the design lifecycle
- Risk ramifications associated with the failure to implement safety-critical design requirements or the implementation of incorrect requirements.

C.9.2 Quality Assurance Plan or Equivalent

“Without exception, the second most important goal must be product quality” [STSC 1994]. Implied from this initial quote, one would assume that safety is an important goal of a program. A quality process is ensured by strict adherence to a systems engineering approach for development of both hardware and software systems. Quality assurance is a planned and systematic set of actions required to provide confidence that adequate technical requirements are established, products and services conform to established technical requirements, and satisfactory performance is achieved. QA includes the qualitative and quantitative degree of excellence in a product. This can only be achieved if there is excellence in the process to produce the product.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The Quality Assurance Plan identifies the processes and process improvement methodologies for development activities. The QAP focuses on requirements identification and implementation (not design solutions), design activities to meet design specifications and requirements, testing to verify requirements, and maintenance and support of the produced product. Safety input to the QAP must focus on the integration of safety into the definition of quality for the product to be produced. Safety must become a function of product quality. The safety manager must integrate safety requirement definitions, implementation, tests, and verification methods and processes them into the quality improvement goals for the program. This includes any software safety certifications required by the customer.

C.9.3 Reliability Engineering Plan

System safety hazards analysis is heavily influenced by reliability engineering data. A sound reliability engineering activity can produce information regarding component failure frequency, design redundancy, sneak circuits, and subsystem and system failure modes. This information has safety impact on design. The REP describes the planning, process, methods, and scope of the planned reliability effort to be performed on the program. Dependent on the scope (breadth and depth) of the SSP, much of the reliability data produced must be introduced and integrated into the system safety analysis. An example is the specific failure modes of a subsystem, the components (whose failure causes the failure mode), and the criticality of the failure consequence. This information assists in the establishment and refinement of the hazard analysis and can produce the information required for the determination of probability of occurrence for a hazard. Without reliability data, the determination of probability becomes very subjective.

The safety manager must determine whether a sufficient reliability effort is in place to produce the information required for the system safety effort. If the development effort requires a strict quantification of hazard risk, there must be sufficient component failure data available to meet the scope objectives of the safety program. Numerous research and development projects produce unique (one of a kind) components. If this is the case, reliability engineers can produce forecasts and reliability predictions of failures and failure mechanisms (based on similar components, vendor data, qualification testing, and modeling techniques) which supplement safety information and increase the fidelity of the safety analysis.

Within the discipline of software safety, the REP must sufficiently address customer specifications regarding the failure reliability associated with safety-critical or safety-related software code. The plan must address:

- Specific code that will require statistical testing to meet user or system specifications
- Any perceived limitations of statistical testing for software code
- Required information to perform statistical testing
- Methods of performing the statistical tests required to meet defined confidence levels
- Specific test requirements
- Test design and implementation

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Test execution and test evaluation.

C.9.4 Software Development Plan

Software-specific safety requirements have little hope of being implemented in the software design if the software developers do not understand the rationale for safety input to the software development process. Safety managers must communicate and educate the software development team on the methods and processes that produce safety requirements for the software programmers and testers. Given that most software developers were not taught that a safety interface was important for a software development program, this activity becomes heavily dependent on personal salesmanship. The software development team must be sold on the utility, benefit, and logic of safety-producing requirements for the design effort. This can only be accomplished if the software development team is familiar with the system safety engineering process.

MIL-STD-498 introduced the software development team to the system safety interface. This interface must be communicated and matured in the SDP. The SDP (usually submitted in draft form with the offeror's RFP response) is the key software document reflecting the offeror's overall software development approach. The SDP must include resources; organization; schedules; risk identification and management; data rights; metrics; quality assurance; control of non-deliverable computer resources; and identification of COTS, reuse, and Government-furnished software the offeror intends to use. SDP quality and attention to detail is a major source selection evaluation criterion.

A well structured and highly disciplined software development process and software engineering methodology help facilitate the development of safer software. This is the direct result of sound engineering practices. The development of software that specifically meets the safety goals and objectives of a particular design effort must be supplemented with system safety requirements that eliminate or control system hazards and failure modes caused by (or influenced by) software. The SDP must describe the software development/system safety interface and the implementation, test, and verification methods associated with safety-specific software requirements. This includes the methodology of implementing generic safety requirements, guidelines (see Appendix E), and derived safety requirements from hazard analyses. The plan must address the methodology and design, code, and test protocols associated with safety-critical software, safety-related software, or lower safety risk modules of code. This defined methodology must be in concert with methods and processes identified and described in the SEMP, SSPP, and SwSPP.

C.9.5 Systems Engineering Management Plan

System safety engineering is an integral part of the systems engineering function. The processes and products of the system safety program must be an integrated subset of the systems engineering effort.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The SEMP is the basic document governing the systems engineering effort. The SEMP is a concise, top-level technical management plan consisting of system engineering management and the systems engineering process. The purpose of the SEMP is to make visible the organization, direction, control mechanisms, and personnel for the attainment of cost, performance, and schedule objectives. The SEMP should contain the engineering management procedures and practices of the developer, the definition of system and subsystem integration requirements and interfaces, and the relationships between engineering disciplines and specialties. The SEMP should reflect the tailoring of documentation and technical activities to meet specific program requirements and objectives.

A further breakdown of the SEMP contents includes:

1. **Technical program planning and control**

    - Program risk analysis
    - Engineering program integration
    - Contract work breakdown
    - Assessment of responsibility
    - Program reviews
    - Technical Design Reviews
    - Technical performance measurement.

2. **Systems engineering process**

    - Functional analysis
    - Requirements allocation
    - Trade studies
    - Design optimization and effective analysis
    - Synthesis
    - Technical interface compatibility
    - Logistics support analysis
    - Producibility analysis
    - Generation of specifications
    - Other systems engineering tasks.

3. **Engineering specialties and integration of requirements**

    - Reliability
    - Maintainability

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Human engineering
- System safety
- Standardization
- Survivability and vulnerability
- Electromagnetic compatibility
- Electromagnetic pulse hardening
- Integrated logistics support
- Computer resources lifecycle management
- Producibility
- Other engineering specialty requirements.

The SEMP must define the interfaces between systems, design, and system safety engineering (including software safety). There must be an agreement between engineering disciplines on the methods and processes that identify, document, track, trace, test, and verify subsystem and system requirements to meet system and user specifications. The SEMP must describe how requirements will be categorized. From a safety engineering perspective, this includes the categorization of safety-critical requirements and the tracking, design, test, and verification methods for ensuring that these requirements are implemented in the system design. The intent of the requirements must be sufficiently incorporated in the design. The lead systems engineer must assist and support the safety engineering and software engineering interface to ensure that the hardware and software designs meet safety, reliability, and quality requirements.

C.9.6 Test and Evaluation Master Plan

Ensuring that safety is an integral part of the test process is a function that should be thoroughly defined in the TEMP. There are three specific aspects of safety that must be addressed.

First, the test team must consider and implement test constraints, bounds, or limitations based on the safety risks identified by the hazards analysis. Test personnel and test management must be fully informed regarding the safety risk they assume/accept during pre-test, test, and post-test activities.

Second, a specific safety assessment is required for the testing to be accomplished. This assessment/analysis includes the hazards associated with the test environment, the man/machine/environment interfaces, personnel familiarization with the product, and the resolution of hazards (in real-time) that are identified by the test team which were not identified or documented in design. The pre-test assessment should also identify emergency back-out procedures, GO/NO-GO criteria, and emergency response plans. The assessment should also identify personal observation limitations and criteria to minimize hazardous exposure to test team personnel or observers.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Third, the test activities include specific objectives to verify safety requirements identified in the design hazard analysis and provided in the generic requirements and guidelines documentation. The safety engineer must ensure that test activities and objectives reflect the necessary requirement verification methods to demonstrate hazard mitigation or control to levels of acceptable risk defined in the analysis. All safety requirement activities and test results must be formally documented in the hazard record for closure verification.

C.9.7 Software Test Plan

The Software Test Plan addresses the software developer's approach and methods for testing. This includes necessary resources, organization, and test strategies. Software development includes new software development, software modifications, reuse, re-engineering, maintenance, and all other activities resulting in software products. The STP must include the schedule and system integration test requirements. DID DI-IPSC-81427 describes the contents and format of the STP. It should be noted that within the contents of the DID, the test developer must describe the method or approach for handling safety-critical (or safety-related) requirements in the STP. The software safety engineering input to the STP should assist in the development of this specific approach. This is required to adjust software safety assessments, schedules, resources, and delivery of safety test procedures.

STP review(s) to support the development of the STP should commence no later than PDR to facilitate early planning for the project.

Software safety inputs to the STP must include:

- LOR table requirements for testing
- Safety inputs to testing requirements (especially those relating to safety-specific requirements), including test bounds, assumptions, limitations, normal/abnormal inputs, and expected/anticipated results
- Safety participation in pre-test, test, and post-test reviews
- Requirements for IV&V and regression testing
- Acceptance criteria to meet safety goals, objectives, and requirements.

C.9.8 Software Installation Plan

The Software Installation Plan (SIP) addresses the installation of the developed software at the user site. This plan should include the conversion from any existing system, site preparation, and the training requirements and materials for the user. There is minimum safety interface with the development of this plan, except in the area of safety-significant training requirements.

Specific safety training is inherent in controlling residual risk not controlled by design activities. Safety risk that will be controlled by training must be delineated in the SIP. In addition, specific

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

safety inputs should be part of regular field upgrades where safety interlocks, warnings, training, or other features have been changed. This is especially true in programs that provide annual updates.

C.9.9 Software Transition Plan

The Software Transition Plan identifies the hardware, software, and other resources required for deliverable support of the software product. The plan describes the developer's plan for the smooth transition from the developer to the support agency or contractor. Included in this transition is the delivery of the necessary tools, analysis, and information required to support the delivered software. From a safety perspective, the developer has the responsibility to identify all software design, code, and test activities in the development process that have safety implications. This includes the analysis that identified the hazards and failure modes that were influenced or caused by software. The transition package should include the hazard analysis and the hazard tracking database that documented the software-specific requirements and traced them to both the affect module(s) of code and to the hazard or failure mode that derived the requirement. Failure to deliver this information during the transition process can introduce unidentified hazards, failure modes, and safety risk at the time of software upgrades, modifications, or requirement changes. This is particularly important if the code is identified as safety critical or becomes safety critical due to the proposed change. Appendix C.11 includes additional information about this issue.

C.10 Hardware and Human Interface Requirements

C.10.1 Interface Requirements

The interface analysis is a vital part of the SHA. This analysis is vital to ensure that software requirements are not circumvented by other subsystems, or within its own subsystem, by other components.

The software interfaces will be traced into, from, and within the safety-significant software functions of a subsystem. These interfaces will be analyzed for possible hazards, and the summary of these interfaces and their interaction or any safety function shall be assessed. Interface addresses of safety-significant functions will be listed and searched to identify access to and from non-safety-significant functions and shared resources.

Interfaces to be analyzed include functional, physical (including the Human/Machine Interface (HMI)), and zonal. Typical hazard analysis activities associated with the accomplishment of a SHA include functional and physical interface analysis. Zonal interfaces (especially on aircraft design) can also become safety critical. Using aircraft designs as an example, there exists the potential for hazards in the zonal interfaces. These zones include fire compartments, dry-bay compartments, engine compartments, fuel storage compartments, avionics bay, cockpit, etc. Certain conditions that are considered hazardous in one zone may not be hazardous in another.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The SHA activity must ensure that each functional, physical, and zonal interface is analyzed and that the hazards are documented. Requirements derived from this analysis are then documented in the hazard record and communicated to the design engineering team.

Before beginning this task, definitions regarding the processes using an interface must be identified. This should include the information passing that interface which affects safety. The definitions of processes are similar to most human factor studies. An example is the Critical Task Analysis (CTA). The CTA assesses which data passes the interface that is critical to the task. Once the data is identified, the hardware that presents the data and the software that conditions the hardware and transmission are examined.

It is recommended that as much preliminary interface analysis (as practical) be accomplished as early in the development lifecycle as possible. This allows preliminary design considerations to be assessed early in the design phase. Requirements are less costly and more readily accepted by the design team if identified early in the design phases of the program. Although many interfaces are not identified in the early stages of design, the safety engineer can recommend preliminary considerations if these interfaces are given consideration during PHA and SSHA activities. From a software safety perspective, interfaces between software, hardware, and the operator will (most likely) contain the highest safety risk potential. These interfaces must be thoroughly analyzed and safety risk minimized.

C.10.2 Operations and Support Requirements

Requirements to minimize the safety risk potential are identified during the accomplishment of the O&SHA. These requirements are identified as they apply to the operations, support, and maintenance activities associated with the system. The requirements are usually categorized in terms of protective equipment, safety and warning devices, and procedures and training. At this later phase in the development lifecycle, it is difficult to initiate system design changes to eliminate a potential hazard. If this is an option, it would be a formal configuration change and requires an ECP approved by the Configuration Control Board (CCB). If the hazard is serious enough, an ECP is a viable option. However, as previously noted, formal changes to a “frozen” design are expensive to the program.

Those hazards identified by the O&SHA and hazards previously identified (and not completely eliminated by design) by the PHA, SSHA, SHA, and HHA are risk-minimized to a lower RAC by protective equipment, safety warning devices, and procedures and training. A high percentage of these safety requirements affect the operator and the maintainer (the HMI).

C.10.3 Safety and Warning Device Requirements

Safety devices and warning devices used within the system and in test, operation, and maintenance activities must be identified during the hazard analysis process. As with the requirements identified for protective equipment, procedures, and training, these requirements

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

should be identified during the concept exploration phase of the program and refined and finalized during the final phases of design. If the original hazard cannot be eliminated or controlled by design changes, safety and warning devices are considered. This should minimize the safety risk to acceptable levels of the RAC matrix.

C.10.4 Protective Equipment Requirements

Another function of the O&SHA is to identify special protective equipment requirements for personnel (test, operator, or maintainer), the equipment and physical resources, and the environment. This can be as extensive as a complicated piece of equipment for a test cell, and as simple as a respirator for an operator. Each hazard identified in the database should be analyzed for the purpose of further controlling safety risk to the extent feasible with the resources available. The control of safety risk should meet programmatic, technical, and safety goals established in the planning phases of the program.

C.10.5 Procedures and Training Requirements

The implementation (setup), test, operation, maintenance, and support of a system require system-specific procedures and training for personnel associated with each activity. Environmental, safety, and health issues for personnel and the protection of physical program and natural resources facilitate the necessity for safety requirements. Safety requirements that influence the system test, operation, maintenance, and support procedures and personnel training can be derived as early as the PHA and be carried forward for resolution and verification in the O&SHA. The safety analyst (with the test and ILS personnel) is responsible for incorporating the necessary safety inputs to procedures and training documentation.

C.11 Managing Change

The adage “nothing is constant except change” applies to software after the system is developed. Problems encountered during system-level IV&V and operational testing account for a small percentage of the overall changes. Problems or errors found by the user account for an additional percentage. The largest numbers of changes are the result of upgrades, updates, and pre-planned (or unplanned) product enhancements. In addition to anomaly resolution, change can be the result of an update to functional or physical requirements, changes in the concept of operations, and technology insertion/refresh. Managing change from a safety perspective requires that the SSS team assesses the potential impact of the change to the system. If the change is to correct an identified safety anomaly or the change potentially impacts the safety of the system, the software systems safety assessment process must rely on analyses and tests previously conducted.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

C.11.1 Software Defect Resolution

Operational software can fail to function as desired or intended by the users. These faults or failures are captured by the user in a trouble report. These reports are commonly referred to as Software Trouble Reports. These reports document the fault, failure, or defect anomaly of the software in context to the functional configuration (state or mode) of the system. STRs are the first step in software defect resolution.

Figure C-5 depicts the process of software defect resolution from the perspective of correcting safety-significant software. Software safety analysis of changes to deployed software is a mini-cycle of the entire software safety engineering process described in this Handbook. If safety engineering was actively involved in the design and test of the original software and these efforts were adequately documented, the analysis of changes is not difficult. Specific tasks to be accomplished for changes made to software include:

- Acquire all legacy system and software design architecture and test documentation
- Identify physical and functional changes to system by updating the FHA
- Determine whether the SSF and SCF lists have changed. Update the SCFL. Ensure deleted SSFs and SCFs are accounted for in the update of the SAR.
- For new SSFs and SCFs, determine the LOR to be used in the update of the software
- Update the PHA, SSHA, SHA, O&SHA as appropriate. Ensure that new hazards, failure modes, and causal factors are identified and documented. Ensure hazard failure modes and causal factors that have been eliminated or controlled differently from the original system are accounted for and documented.
- Update the SRA by updating the GSSR, CSSR, and MSSR lists. Ensure that new requirements are identified and documented in the RTM and specifications.
- Assist in the development of software test scenarios, tests, and procedures. Ensure regression testing is accomplished in accordance with the LOR table.
- Update the SAR by accounting for new or deleted hazards, failure modes, or causal factors.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

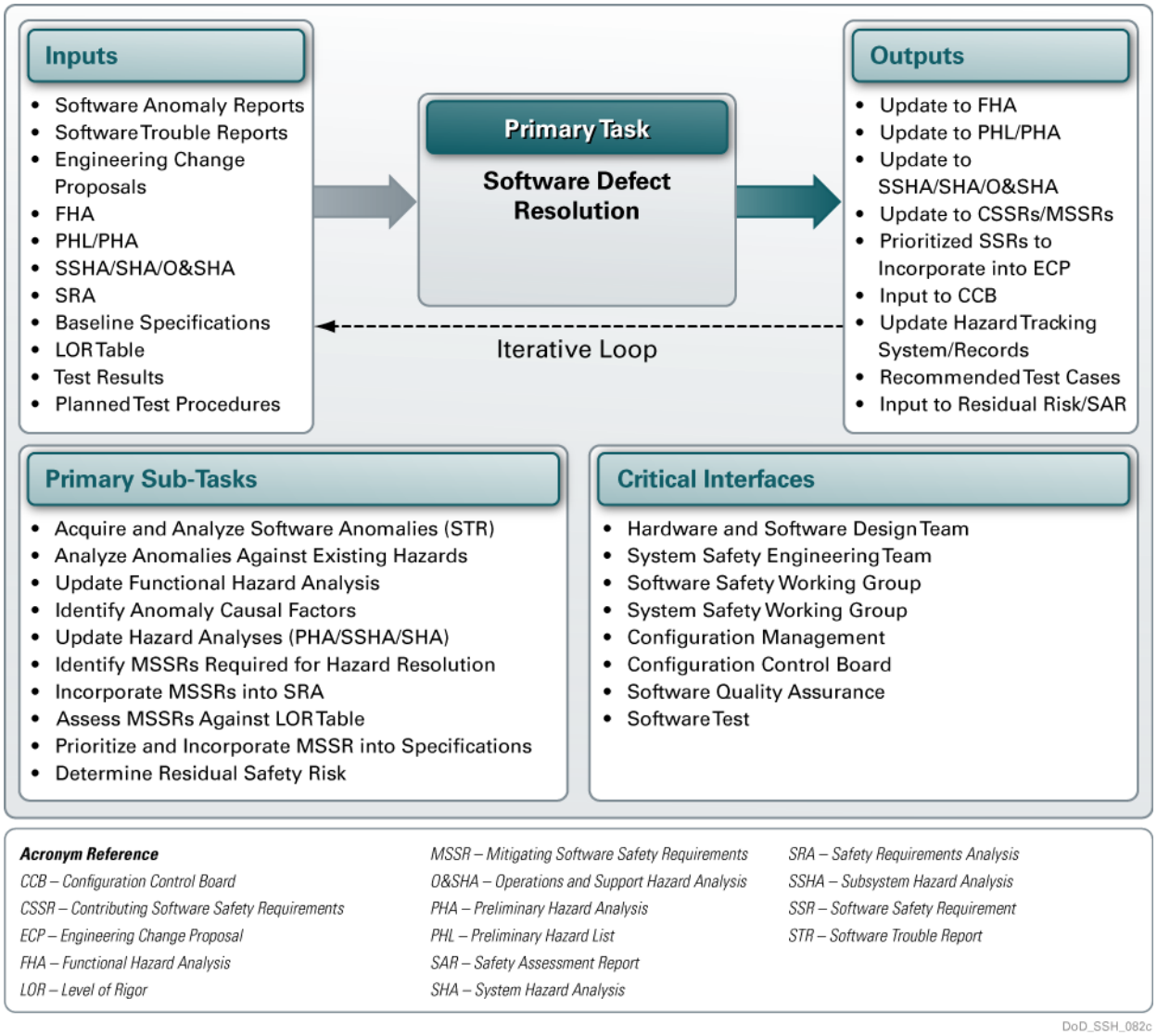

Figure C-5: Software Defect Resolution

C.11.2 Technology Insertion and Refresh

As with software defect resolution, updates to the existing system can functionally or physically change the system. Updates to the system can be to resolve defects, update the functionality of the system, and update the environments in which the system will operate. One form of update to the system can be a technology refresh or insertion (Figure C-6). Regardless of the form of the update, the software safety engineering effort follows the processes described in the Handbook. In the case of technology insertion and refresh, the original design team should adequately account for the planned update or insertion of new technology in the system. This will minimize the safety impact of the technology insertion.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

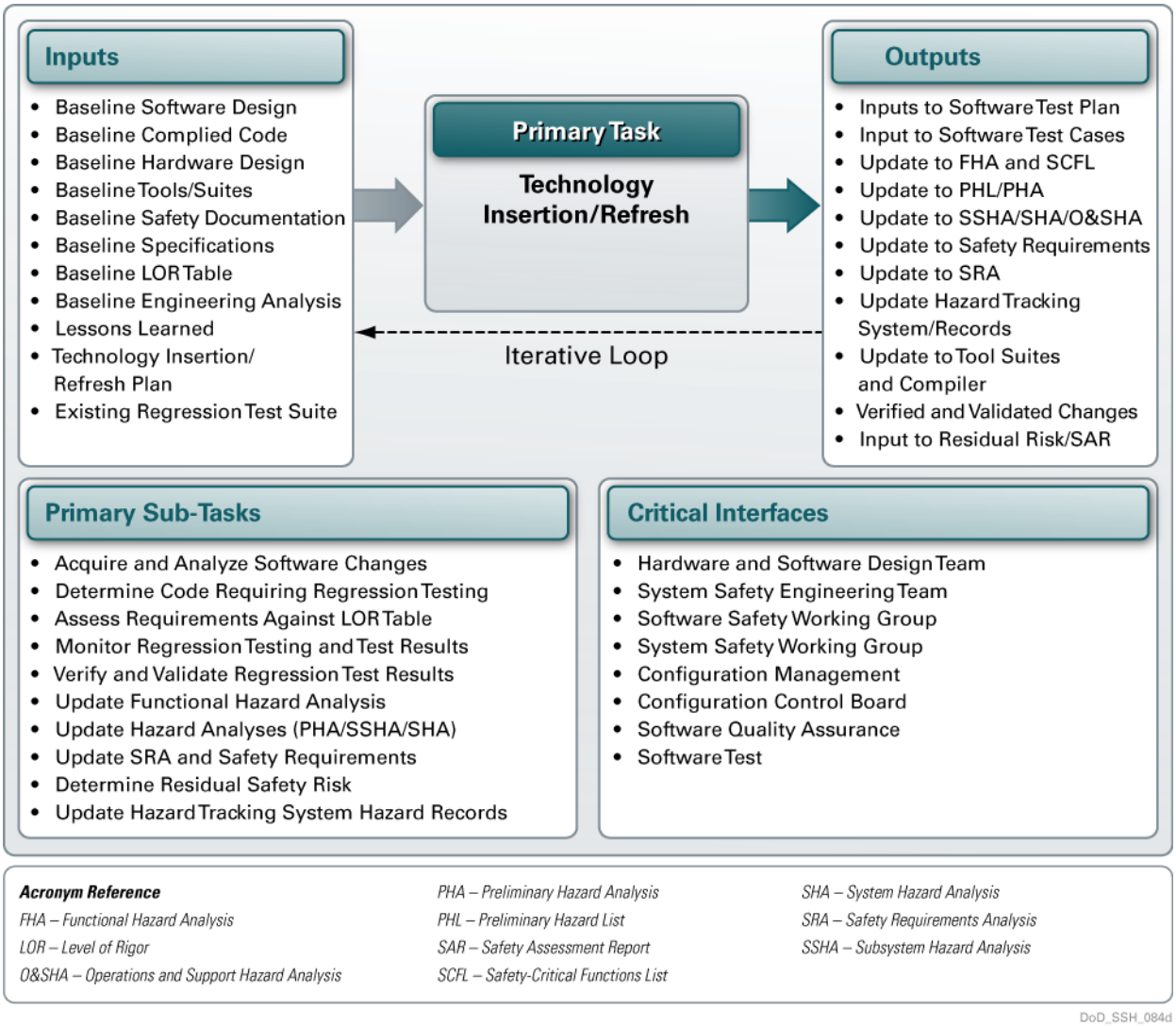

Figure C-6: Technology Insertion and Refresh

As with defect resolution, the technology insertion and refresh effort will:

- Acquire the original Technology Insertion/Refresh Plan, if it exists
- Acquire all legacy system and software design architecture and test documentation
- Identify physical and functional changes to system by updating the FHA
- Determine whether the SSF and SCF lists have changed. Update the SCFL. Ensure deleted SSFs or SCFs are accounted for in the update of the SAR.
- For new SSFs and SCFs, determine the LOR to be used in the update of the software

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Update the PHA, SSHA, SHA, O&SHA as appropriate. Ensure that new hazards, failure modes, and causal factors are identified and documented. Ensure hazard failure modes and causal factors that have been eliminated or controlled differently from the original system are accounted for and documented.
- Update the SRA by updating the GSSR, CSSR, and MSSR lists. Ensure that new requirements are identified and documented in the RTM and specifications.
- Assist in the development of software test scenarios, tests, and procedures. Ensure regression testing is accomplished in accordance with the LOR table.
- Update the SAR by accounting for new or deleted hazards, failure modes, or causal factors.

C.11.3 Software Configuration Control Board

CM is a system management function wherein the system is divided into manageable physical or functional configurations and grouped into CIs. The CCB controls the design process through the use of management methods and techniques, including identification, control, status accounting, and auditing. CM (Figure C-7) of the development process and products within that process is established once the system has been divided into functional or physical configuration items. The CCB divides any changes into their appropriate classes (Class I or Class II) and ensures that the proper procedures are followed. The purpose of this function is to ensure that project risk is not increased by the introduction of changes by unauthorized, uncontrolled, poorly coordinated, or improper changes. These changes could directly or indirectly affect system safety, and therefore require verification, validation, and assessment scrutiny.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

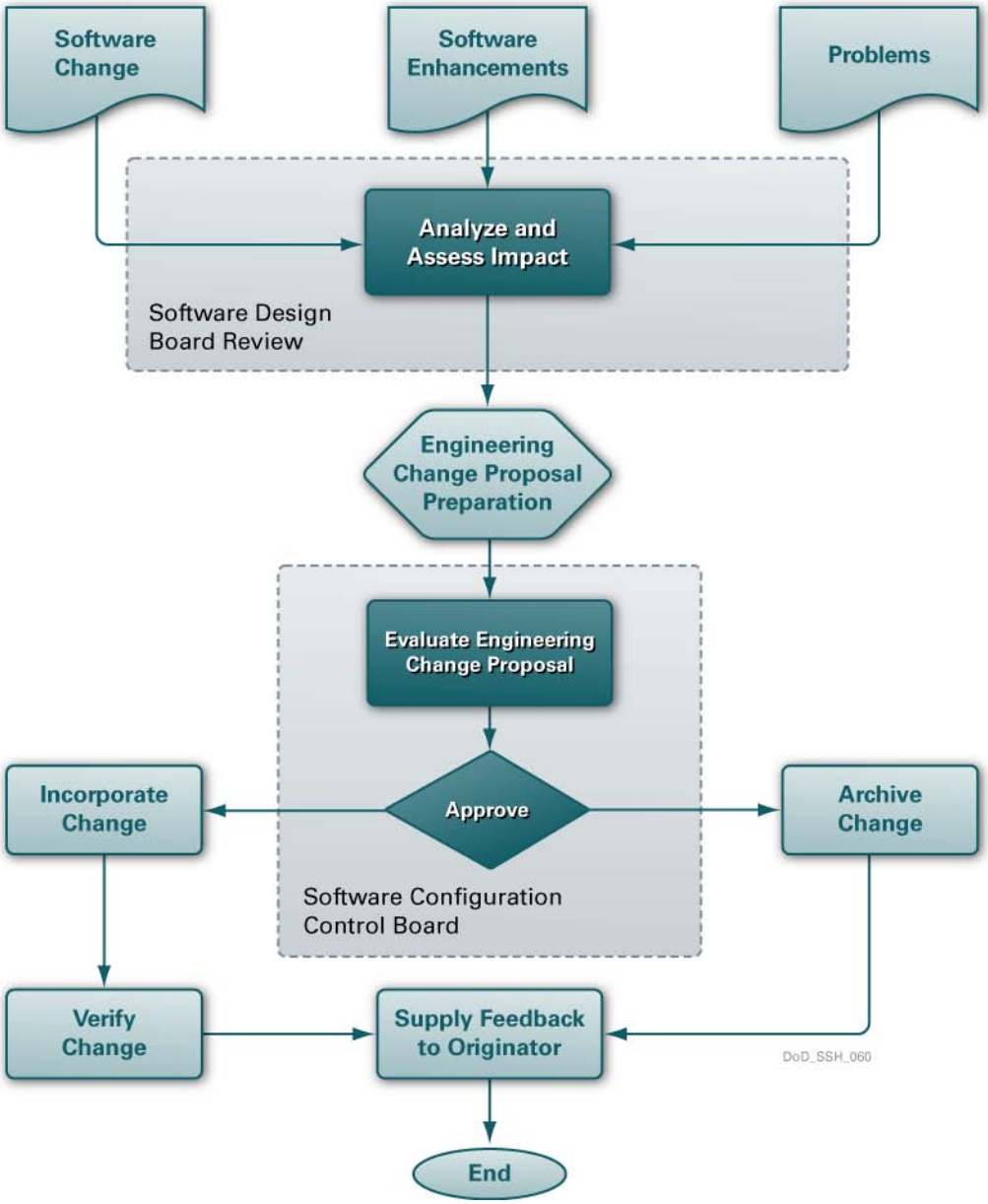

Figure C-7: Generic Software Configuration Change Process

The CCB assists the Program Manager, design engineer, support engineers, and other acquisition personnel in the control and implementation of Class I and Class II changes. Class I changes are those which affect form, fit, or function and require user concurrence prior to developer implementation. Class II changes are those not classified as Class I. Examples of Class II changes include editorial changes in documentation or material selection changes in hardware.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The CCB ensures that the proper procedures for authorized changes to the CI or related products or interfaces are followed and that risk is not increased by the change. The CCB should also ensure that any intermediate steps that may halt and expose the project to increased safety risk while halted are controlled. The system safety assessment regarding a configuration change must include:

- Thorough review of the proposed change package ECP prepared by the engineer responsible for the change
- Effects of the proposed change on subsystem and system hazards previously identified, including existing and new functional, physical, or zonal interfaces
- Determination as to whether the proposed change introduces new hazards to the system or to its operation and support functions
- Determination as to whether the proposed change circumvents existing (or proposed) safety systems
- Analysis of all hardware/software and system/operator interfaces.

The SSS team follows the same process (on a smaller scale) they followed during system development. The analyses will have to be updated and appropriate tests re-accomplished, particularly the safety tests related to those portions of the software being modified. The development of the change follows the ECP process, including the configuration control process. The overall process follows through and concludes with a final safety assessment of the revised product. It is important to remember that some revalidation of the safety of the entire system may be required depending on the extent of the change.

One important aspect of managing change is the change in system functionality. This includes the addition of new functionality to a system that adds safety-critical functions to the software. For example, if the software developed for a system did not contain safety-critical functions in the original design, yet the modifications add new functionality that is safety critical, the software safety effort will have to revisit a great deal of the original software design to assess its safety risk potential. The software safety analysts will have to revisit both the generic safety design requirements and the functionally derived safety requirements to determine their applicability in light of the proposed software change. Where the tailoring process determined that certain generic requirements were not applicable, the rationale will have to be examined and the applicability re-determined. The PA may argue that the legacy software is safe and so the new functionality requires that it be the only portion examined. Unless software engineering standards are rigorously followed during the original development, it will be difficult for the safety analyst to ensure that the legacy software cannot adversely impact the new functionality. The process used is the same as it was for the original software development.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

APPENDIX D COTS AND NON-DEVELOPMENTAL ITEM SOFTWARE

D.1 Introduction

The safety assessment of COTS and NDI software poses one of the greatest challenges to the safety assessment and ultimate acceptance or certification of safety-critical and mission-critical systems. It is commonplace for design and development teams to select COTS and NDI software as cost or time saving solutions for programs. However, COTS and NDI software pose a potential safety risk to these programs based on the specific and unique functional characteristics of the software when integrated into the system. The system safety team must ensure that all safety risk is identified and accounted for in the risk assessment reports and the specific safety cases associated with the system within its defined operational environments. While the main body of the Handbook describes the typical software safety engineering process, this appendix focuses on the unique tasks of selecting and integrating COTS/NDI software. Within this appendix, the term COTS will be used exclusively, but will infer the possibility of NDI software as well as COTS software. This Appendix will provide information and guidance regarding:

- D.2 – The general characteristics of COTS software, including the advantages and disadvantages of its use
- D.3 – The activities required to make an official determination whether COTS software is appropriate for a specific system or system application
- D.4 – Implementing an example COTS software safety selection process
- D.5 – Safely integrating COTS (after selection) into the system
- D.6 – Safety risk reduction methods
- D.7 – The COTS safety case
- D.8 – Summary.

D.2 Characteristics of COTS Software

COTS software is developed for a wide variety of applications in both the Government and commercial marketplace. COTS developers may use an internal company or industry standard(s) for software development activities, or they may develop to Government acceptance or certification criteria. Regardless of the development and test environments, it is commonplace for COTS/NDI software vendors to only release compiled versions of software with limited documentation. In most cases, these limitations make it difficult to understand the vendor's software, respective problems or issues, and how it will ultimately impact the safety of the system where it will be integrated.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Each particular COTS application has its own unique set of inherent attributes that can be described as either advantages or disadvantages for particular use within a specific application or system. The advantages and disadvantages of a COTS item must be carefully weighed against the program requirements before determining COTS usage. PMs must understand that the use of COTS items often appears to be the cheaper alternative because standard DoD development tasks have not been performed. As the costs to perform the necessary tasks to fully evaluate the COTS items in the system application are considered, the option to use COTS may not be the best alternative. Specifically, hazards must be identified, risks assessed, and the risk made acceptable regardless of how the component/function is developed. The decision to use COTS items does not negate system safety requirements.

D.2.1 Advantages of COTS Software

The potential advantages of using or considering COTS software include:

- Cost savings (no development costs)
- Rapid insertion of new technology
- Proven product/process
- Possible broad user base
- Potential technical support
- Potential logistics support.

Each of these possible advantages should be weighed against the potential disadvantages and safety risk potential for use of COTS software applications.

D.2.2 Disadvantages of COTS Software

The potential disadvantages of using or considering COTS software include:

- Potential for limited development, test, or configuration control documentation
- Unknown development history (standards, quality assurance, test, analysis, failure history, etc.)
- Unavailability of design and test data (drawings, test cases and procedures, test results, etc.)
- Proprietary design prohibitions
- Unable to modify based on limited proprietary or data rights
- Unknown functionality and functional limitations (operational, environmental, stress, etc.)
- Limited or no supportability from the developer or vendors (configuration control, tech support, updates, etc.)

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Unnecessary functionality or capabilities (the potential of "hidden" or undocumented functionality)
- Potential obsolescence of the COTS application
- May not be developed to best industry or Government practices or certification criteria
- Unavailability of safety analyses for the COTS application
- Potential for increased test and analysis required for safety verification, safety release, or safety certification
- Potential need for periodic updates and the unknown impact of those updates
- Functions or tasks unneeded by the intended program
- Unable to modify due to licensing requirements, or the purchase of the license agreement is not practical for the program.

D.3 Making a COTS Use Decision

The COTS software selection process guidance establishes the basic criteria for selecting and using COTS applications for safety-critical systems, including their architecture and design. The process provides guidance for evaluating COTS in the context of a system design with its mishap and hazard risk potential. The guidance can apply to new designs and modifications, as well as to existing designs undergoing COTS upgrades. Figure D-1, along with the advantages and disadvantages defined in D.2.1 and D.2.2, identifies three initial areas of consideration to begin the COTS decision-making process.

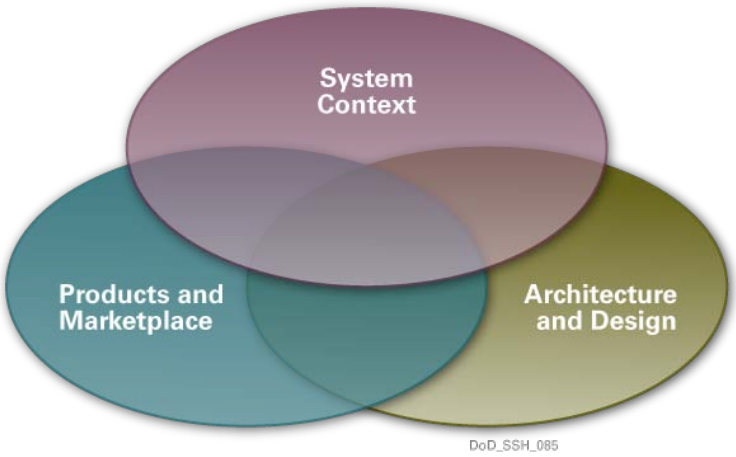

Figure D-1: Initial Considerations for COTS Selection

Within the context of the system and its intended use environments, a fundamental question must be answered. Should COTS software be allowed for use within safety-critical applications? If

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

the answer to this question is affirmative, selecting the most correct application with the least safety risk potential is imperative. The combination of the COTS design architecture, the product marketplace, and the context of the system must be evaluated as to whether the COTS application makes logical sense within a safety-critical system.

The selection process becomes a matching exercise where the requirements for the subsystem components are matched with the functionality of the COTS application. A match consists of functional and performance specifications matching the system requirements and any COTS requirements developed by the program's system safety engineer. Components that most closely meet the functional, performance, and safety requirements are considered the best match. Once the functional matches are accomplished, the COTS selection process should be evaluated against three important criteria for the potential of COTS software performing within the design architecture of the host system. These three evaluation criteria are depicted in Figure D-2.

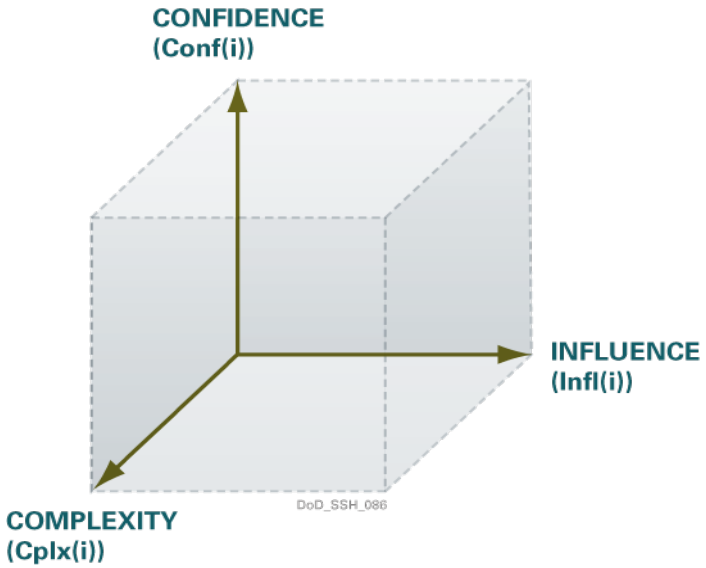

Figure D-2: COTS Evaluation Space

D.3.1 Confidence

Webster's Dictionary defines confidence as "the quality or state of being certain." The COTS evaluation team must possess the confidence or certainty that the software application selected will perform the intended functions within the intended operational environments and system(s) in which it will be used. To possess this confidence, the COTS item must be thoroughly evaluated from both a legacy and future perspective.

The operational environment for which the COTS item was originally designed must be known in order to determine if it can be successfully and safely integrated within the context of the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

targeted system operational environment. Each COTS item has a unique development history which must be thoroughly evaluated to determine its effect on the safety of the new system. Selection of the COTS item should follow a structured evaluation process with respect to its intended operational environment and CONOPS. The COTS item may provide more capability than desired (environmental, operational, performance, or reliability) or less capability than required. The system developer must know and include the effects of integrating the COTS item into the system when it has a different set of capabilities than required. The end result of these efforts will determine if the costs and risks are greater than implementing other recommendations for reducing the risk of safety-critical applications of COTS products. To make this determination, the system engineer must be cognizant of the following confidence criteria:

a. Adaptability of the COTS item to the targeted system, including the physical architecture, software architecture, functional architecture, functional and physical interfaces, and requirements compatibility
b. Government or commercial use similarities, differences, and precedence
c. COTS item vendor support, including operation manuals, training documentation, upgrade history, change notifications, supportability costs, and configuration management artifacts
d. Accessibility of COTS item documentation, including:
    - Design standards and guidance used in development and test
    - Design drawings and specifications
    - Safety analyses accomplished
    - Test plan procedures and associated test results
    - Failure or anomaly reports.

Most COTS items are in use by other systems, which means they already have field experience. The use history of a COTS item will provide some clues for safety, reliability, maintainability, and support of the item. The critical variable is how much data or information on the COTS item is available and whether it has applicability to its use in the targeted system.

In most cases, COTS items are built to commercial standards, but the particular standards followed can vary or be unknown. Information on all of the standards that were used in the development and manufacture of the COTS item is relevant for system integration.

Occasionally, system developers may purchase the required documentation from the COTS developer who often charges premium prices for it. However, its availability provides the safety team with detailed knowledge of the design to identify hazard causal factors and design specific interlocks in the applications software to preclude their occurrence.

D.3.2 Influence

Webster's Dictionary defines influence as "to have an effect on the condition or development of…" By understanding the technical nature of the impact that the COTS product imparts upon

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

the targeted system, the evaluator is able to characterize the safety effort required to analyze, test, and eventually identify residual risk within the system. The safety impact of COTS on the target system lends itself to the influence that the application possesses.

In the conduct of a system safety program, the safety team analyzes the system to identify the undesired events (mishaps), the hazards that provide the preconditions for the mishap, and the potential causal factors that may lead to those hazards. By eliminating or reducing the probability of occurrence of the causal factors, the analyst reduces the probability of the hazard and the overall mishap risk associated with the system. As noted earlier, the safety analyst requires detailed knowledge of the system design and its intended use to eliminate or reduce the risk of hazards.

COTS software poses several issues in this regard. Generally, vendors provide only user documentation from commercial software developers. The type of documentation necessary to conduct detailed analyses is usually not available or is difficult to obtain. The developer may not even generate high-level specifications, functional flow diagrams, data flow diagrams, or detailed design documents for a given commercial software package. The lack of detailed documentation limits the software safety engineer to identifying system hazard causal factors related to the COTS software at a high level. The safety analyst has limited information to develop and implement design requirements or analyze the design implementation within the COTS software to eliminate or reduce the probability of occurrence of these causal factors. However, the lack of design detail or safety engineering artifacts for the proposed COTS application cannot be interpreted as a rationale to do nothing. In fact, it should be interpreted as the impetus to analyze the proposed COTS application against the functional and operational environments of the target system as thoroughly as possible, albeit a “black box” analysis.

The evaluation of the influence a COTS product may have within the targeted system at milestone events during the acquisition process can be characterized with a six-step process. NOTE: Section 4 of the Handbook fully describes the process for evaluating software within the system and safety context. The six steps defined here are considered the minimum activities required to evaluate a proposed COTS application in the context of the target system. Refer to Section 4 for more detail to accomplish these tasks.

- Conduct/update a Functional Hazard Analysis to define the safety-significant functionality associated with the integration of the COTS product into the targeted system. The products of this task should list safety-critical and safety-related functions that the proposed COTS application will perform for the target system and list the safety requirements to be included in the specifications. The safety requirements will be MSSRs, as defined in Section 4, to specifically mitigate defined safety risk.
- Assign a Software Control Category for each safety-significant function (e.g., autonomous, semi-autonomous, redundant fault tolerant, or no safety impact). The information contained in Table 4-1 provides a methodology for assessing the criticality of the system’s safety-significant functions and the software’s control

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

capability in the context of the software's ability to implement the functions. As stated in Section 4, each of the software safety-significant functions can be labeled with an SCC for the purpose of defining the level of rigor that will be required in the function's design, implementation, test, and verification.

- Assign the safety-significant functions a Software Criticality Index. As reflected in Table 4-2, the SCI is a mechanism to assess software criticality impact on the system in the event of a failure and, based on command, control, and autonomy authority for a specific safety-significant function, determine the LOR required in software development and test activities to ensure safety assurance and integrity within the system context.
- Define the tailored LOR based on the acceptance or certification criteria of the customer. Figure 4-13 of the Handbook provides an example level of rigor template.
- Develop the technical engineering evidence that supports the successful completion of the LOR for all COTS safety-significant functions. The engineering evidence must also include requirements traceability of all MSSRs defined to reduce the safety risk potential of the COTS application.
- Define the inherent risk of not accomplishing the tasks indicated in the LOR table. The risk of not completing the required analyses or tests could be due to limitations within the design, schedule, or cost associated with integrating the COTS product within the targeted system.

D.3.3 Complexity

Webster's Dictionary defines complexity as "something made up of or involving an often intricate combination of elements…" System complexity can be visualized from different perspectives by different stakeholders throughout the design and development process. In the context of a COTS application, complexity is considered the number of intricate variables that the COTS product represents in the context of the overall target system. Attributes of complexity in this context include:

- The number of software lines of code
- The number of safety-significant functions
- The number of possible paths through the software
- The number and frequency of nesting levels in the software
- The number and complexity of the functional and physical interfaces between the hardware, software, and human elements
- Modularity of the software/hardware architecture in context to the functional and physical interfaces (e.g., separation of safety-significant components)
- Access to the source code
- Availability of coding guidelines used for COTS design and test
- Knowledge of coding restrictions
- Knowledge of software use limitations.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The ability to modify a COTS item is a prime consideration within the selection process and involves the amount of control the system developer has over the COTS item. The first question to be answered is, "Does the COTS item require modification in order to meet system requirements?" The second question to be answered is, "Does the COTS item provide the capability for modification?" If the COTS application can be modified, is this task relegated to the original vendor, or can the system developer modify it? Any modification of a COTS item is a serious consideration because of the costs involved.

The test community may determine complexity based on the ability to verify and validate system requirements given the capacity and capability of the test facilities. Testing of the COTS software is limited in its ability to provide evidence that the software cannot influence system hazards. Testing in a laboratory cannot duplicate every nuance of the operational environment, nor can it duplicate every possible combination of events. Based on knowledge of failures and operational errors of the software design, test engineers can develop procedures to test software paths specifically for safety-critical events. Even when the developer knows the design and implementation of the software in detail, constraints still apply. The testing organization, like the safety organization, must still treat COTS software as a "black box." This includes developing tests to measure the response of the software to input stimulus under (presumably) known system states. Hazards identified through black box testing are sometimes happenstance and are difficult to duplicate. Timing and data senescence issues also are difficult to fully test in the laboratory environment, even for software of a known design. Without detailed knowledge of the design of the software, the system safety and test groups can only develop limited testing to verify the safety and fail-safe features of the system.

D.4 COTS Software Safety Selection Process Example

The system safety team can provide a valuable service to the COTS selection process by providing a structured evaluation approach, including and accounting for the positive and negative attributes of the candidate COTS applications. Using the confidence, influence, and complexity criteria introduced in Section D.3, evaluation metrics can be introduced to the selection process to bring engineering evaluation and clarity to the decision-making process. NOTE: Sections 4.4.1 through 4.4.4 represent an example technique of what can be accomplished. This example should not to be interpreted as a formal requirement or the only approach to be implemented for a COTS selection process.

D.4.1 Confidence Metric

Using the confidence factors (a through d) defined in Section D.3.1, a simple metric table can be produced to represent the confidence the evaluation may possess of a candidate COTS application. Figure D-3 is an example of a metric that can be used to consider the confidence component of the COTS evaluation process. This is only one element of the evaluation criteria.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Confidence Factor (Conf)	Available Confidence Criteria
5	a, b, c, and d
4	a, b, and c
3	a and b
2	a
1	No Criteria Available

DoD_SSH_087a

Figure D-3: Example COTS Confidence Criteria Metric

D.4.2 Influence Metric

Upon completion of a preliminary mishap and hazard analysis and a functional analysis of the candidate COTS application, an influence metric can be produced. Figure D-4 represents an example safety influence metric that can be used in the assessment of how the candidate COTS application influences the safety risk impact to the target system.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Influence Factor (Conf)	Risk Level
5	Software implementation and potential contributions to system-level hazards are considered **LOW** risk without safety verification through high-level safety test.
4	Software implementation and potential contributions to system-level hazards are considered **MEDIUM** risk without adequate safety verification. Safety verification requires high-level design analysis and testing.
3	Software implementation and potential contributions to system-level hazards are considered **SERIOUS** risk without adequate safety verification. Safety verfication requires requirements analysis, design analysis, and in-depth testing.
2	Software implementation and potential contributions to system-level hazards are considered **HIGH** risk without adequate safety verification. Safety verification requires significant requirements analysis, design analysis, code analysis, and testing.
1	Software implementation and potential contributions to system-level hazards are considered **UNACCEPTABLE**.

DoD_SSH_088c

Figure D-4: Example Safety Influence Metric

Note that each of the defined example metrics is used in a total or cumulative evaluation and decision-making activity. Each metric as a stand-alone attribute only provides a portion of the decision-making base of knowledge.

D.4.3 Complexity Metric

The complexity of the candidate COTS application in context to the target system can be somewhat complex itself. Three specific attributes of complexity will be considered in this example complexity metric:

- Safety factors (Figure D-5)
- Testability factors (Figure D-6)
- Integration factors (Figure D-7).

The safety factors of the candidate COTS application must be centered on the functionality of the application in context to the proposed target system of consideration. The mishap and hazard analysis combined with the functional analysis will provide the basis for the safety factors portion of the complexity assessment. Of primary interest are whether the functionality of the COTS application is considered safety significant and the specific software control capabilities

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

are within the system context. This information is analyzed and interpreted within the safety factors (S) metric.

Figure D-5: Example Safety Factors of Complexity

Testability attributes of the proposed COTS application within the system context provide the testability factors (T) of the complexity equation. Testability assesses the potential of understanding of the functionality (or "blackness") of the application and whether specific functional testing can be accomplished to satisfy the integrity attributes of the software.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Testability Factor (T)	Level of Testing
5	Static Code Analysis
4	White Box Testing
3	Grey Box Testing
2	Black Box Testing
1	Requirements for COTS Product are Untestable

DoD_SSH_090a

Figure D-6: Example Testability Factors of Complexity

The last attribute of the complexity equation involves the integration of the proposed COTS application in the context of the target system and whether that system is an intimate element of a larger system-of-systems. The integration factor (I) will factor in the risk of the COTS within the context of use by the target system or other systems in an interoperability environment.

Integration Factor (I)	Level of Integration
5	Stand Alone COTS System
4	COTS System + Multiple Systems
3	(COTS System + Multiple Systems) x Multiple Platforms
2	((COTS System + Multiple Systems) x Multiple Platforms) x Allies
1	Potential System Integrations Are Undefined

DoD_SSH_091a

Figure D-7: Example Integration Factors of Complexity

The complexity factor can now be mathematically derived with the following equation (Cplx):

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

$$Cplx = \frac{S + T + I}{3}$$

The Cplx equation will be used in the final equation for the "measure of success" for the use of COTS software in the defined system and its intended environments.

D.4.4 Measure of Success

By using the three basic metric factors of confidence, influence, and complexity during the COTS selection process, the system safety engineer can predict a measure of success (M_S) percentage from which design, development, and test decisions can be made to influence the overall acquisition process. For instance, using the values established in the previous tables, the M_S equation illustrates the potential M_S percentage as a function of the total volume of the COTS evaluation space depicted in Figure D-2.

$$M_s = \left[\frac{Conf * Infl * Cplx}{125}\right] * 100$$

Metrics described in Section D.4 represent one approach in the evaluation of COTS software applications. This does not infer that it is the only approach.

D.5 COTS/NDI Software Safety Process Implementation

Once the decision is made to incorporate a selected COTS application into a system or SoS, the safety analyst will treat this software as it would any other safety-significant component of the system. This will drive the analyst to follow the normal software safety process and tasks defined in Chapter 4. The difference between the normal software safety processes and a COTS-specific process is the "blackness" of the COTS application within the system context. Minimum software safety process tasks for COTS applications are defined in Section D.3.2. Where the functional attributes of COTS application are black box in terms of documentation and understanding, specific design requirement considerations must be defined to reduce the safety risk.

D.6 Safety Risk Reduction Requirements

Risk reduction in the systems employing COTS varies with the degree of abstraction of the non-development components. Legacy components and developed components, whether they are hardware, software, or firmware, can be used to develop causal controls that can mitigate risk. COTS can also be modified provided adequate documentation is available; however, should this course be chosen, the non-developed software can no longer be considered COTS. Once modified, it is considered developed software and falls under all process requirements and

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

guidelines for developed software and firmware. Should a vendor change their commercial component or product, it remains a COTS product only when the vendor offers the recommended product enhancement into the marketplace for all to purchase. Otherwise, a vendor-modified or third party-modified COTS becomes a newly designed component that must possess a supplier, maintenance plan, and other supporting documentation that would fall under the guidance provided within this appendix.

The system developer has a number of options for addressing the risk associated with the application of COTS software in safety-critical systems. The first option is to treat the COTS as trusted software. This option virtually guarantees that a hazard will occur at some point in the system's lifecycle. The system development group must employ techniques to reduce the risk of COTS software in safety-critical applications. Note that several of the risk reduction technique paragraphs are very similar to those that were presented in Chapter 4.4. They are included here to specifically discuss COT implications.

D.6.1 Applications Software Design

The straightforward approach is to design the application software for any eventuality. However, this is often difficult, particularly if the developers are not aware of the full range of functionality of the COTS software. This approach requires the safety analyst to identify all potential causal factors and ensure that the applications software design will respond in a safe manner. This often adds significant complexity to the software and will likely reduce its availability. The safety analyst must thoroughly evaluate any change to the COTS software and ensure that the applications software changes to mitigate any risks. This process can be expensive.

Another technique related to this issue under investigation by the system safety organization at Lockheed-Martin in Syracuse, New York, is the use of discriminators. Discriminators are characteristics of the COTS product that are potentially safety critical in relation to the applications software. The developer uses discriminators to determine when changes to the COTS software may impact the applications software. This process appears viable; however, in practical application it has proven difficult to implement. One reason for this difficulty is the determination of changes that indirectly impact the safety-critical functionality of the software. The ability of unrelated software to impact the functionality that influences the safety-critical aspects of the applications software depends directly on the quality of the software development process used by the original COTS developer. If the developer maintained high quality software engineering practices, including information hiding, modularization, and weak coupling, the likelihood that unrelated software can impact the functionality that influences safety-critical portions of the applications software is reduced.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

D.6.2 Middleware or Wrappers

An effective method for reducing the risk associated with COTS software applications is to reduce its influence on the safety-critical functions in the system. This requires isolation (e.g., firewalls) of the COTS software from the safety-critical functions. Isolation from an operating system requires a layer of software, often called middleware, between the applications software and the operating system. The middleware interfaces with both the operating system and the applications software. All interactions between the applications software and the OS take place through the middleware. The middleware implementation simplifies the design of the applications software by eliminating the need to provide robustness to change in the operating system interface. The developer can specify and maintain a detailed interface design between the middleware and the applications software. The developer must then provide the necessary robustness for the operating system in the middleware. The middleware also contains the exception and interrupt handlers necessary for safe system operation. The middleware may contain macros and other programs used by numerous modules in the applications software rather than relying on those supplied by the COTS products.

Isolation of safety-critical functions from the COTS software may require application-specific software "wrappers" on either side of the COTS software, effectively isolating the two software packages. This technique is similar in nature to the middleware discussed above, but is more specific to a particular function within the system. The wrapper will perform the necessary sanity checks and isolation to preclude hazard occurrence. The safety analyst must consider the risk associated with COTS during the PHA phase of the program. The identification of potential causal factors associated with COTS helps identify the need for such wrappers. By identifying these causal factors, the analyst can develop design requirements for the wrapper. The safety analyst knows the normal safety-critical software inputs and outputs and the specific bounds of that data. The wrapper prevents anything else from getting through.

The use of isolation techniques has limitations. One limitation occurs when interfaces to external components pass through COTS software. For example, a system may use a hard disk to store and retrieve safety-critical data. The COTS operating system provides the handler and data transfer between the disk and the applications software. For devices such as data storage and retrieval, the developer must ensure that the devices selected for the system provide data integrity checks as part of that interface. Another limitation is the amount of software required to achieve adequate isolation. If the applications software has interactions with the COTS software in numerous functional areas, the complexity of the middleware and the number of wrappers required may make them impractical or cost prohibitive.

Wrappers and middleware are effective techniques for isolating safety-critical functionality. If identified early in the system development process, they are also cost effective although there are costs for the development and testing of this specialized software. Another benefit is that when properly designed, the wrappers and middleware reduce the impact of changes in the COTS products from both the system safety and system maintainability perspective. Wrappers and middleware can be relatively simple programs that the developer can readily modify to

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

accommodate changes in the COTS software. However, as noted above, when the interactions are complex in nature, the middleware and wrappers can become complex as well.

D.6.3 Message Protocol

A technique for isolating safety-critical data from COTS software (operating systems, network handlers, etc.) is to package all communications and data transfers in a robust manner. Specifying a communications protocol that provides unique identification of the message type and validation of the correct receipt of the data transfer will ensure that the COTS products do not adversely affect safety-critical data. The degree of robustness required depends on the criticality of the data. For highly critical data, a message protocol using CRC, Fletcher Checksums, or bit-by-bit comparisons provides a high degree of assurance that safety-critical data passed between system components is correct. Less robust data checks include arithmetic and linear check sums and parity checks coupled with well-defined message structures. Middleware may incorporate the message handler, including the CRC or checksum software, thus offloading that functionality from the applications software. This approach is relatively easy and cost effective if implemented early in the system design. However, like all other aspects of system design, late identification of these requirements results in a significant cost impact.

D.6.4 Designing Around COTS

"Designing around COTS" is a technique often used to control the functionality of a COTS application. Embedding exception and interrupt handlers in the applications software ensures that the application software maintains control of the system. However, it is generally not possible to wrestle control for all exceptions or interrupts from the operating system and environment or the compiler. Micro-code interrupt handlers embedded in the microprocessor BIOS often cannot be supplanted. Attempting to supplant these handlers is likely to be more hazardous than relying on them directly. However, careful attention to the design of the applications software and its interaction with the system when these interrupts occur can mitigate any related hazards. Message protocol and watchdog timers are other examples of designing around COTS software.

The system developer can design the applications software to ensure robustness in its interface with the operating system and environment. The robustness of the design depends on the ability of the analyst to identify potential failure modes and changes to the OS or environment. For complex systems, the time and resources required to identify all of the failure modes and potential changes is very high. The additional complexity in the applications software may also introduce hazard causal factors into the design that were not present in the simpler design. Other aspects of designing around COTS software are beyond the scope of this Appendix. The need for design features depends directly on the COTS software in the system and the interactions between the COTS software and the applications software.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

D.6.5 Analysis and Test of COTS Software

The system developer may have access to detailed design documentation on NDI products. Availability varies from product to product, but is generally expensive. However, the detailed analysis of the NDI software and its interactions with the application software provides the greatest degree of assurance that the entire software package will safely execute in the system context. Analysis of NDI products allows for the development of directed testing of the application software in the NDI environment to determine if identified causal factors will result in an undesired condition. However, it may be difficult to generate failure modes in NDI software without actually inserting modifications. Earlier paragraphs discussed this option and noted that the project team must evaluate the cost of procuring this documentation against the cost of other options. A portion of the decision must consider the potential consequences of a safety-critical failure in the system.

D.6.6 Eliminating Functionality

Eliminating unnecessary functionality from operating systems and environments reduces the risk that these functions will corrupt safety-critical functions in the application software. Some functionality, such as file editors, is undesirable in safety-critical applications. For example, a U.S. Navy program retained an operating system's file editor to allow maintenance personnel to insert and test patches and perform software updates. The users discovered this capability and used it to resolve problems they were having with the system. One of the problem resolutions discovered by the sailors also overrode a safety-interlock in the system that could have inadvertently launched a weapon. Although an inadvertent launch did not occur, the potential for its occurrence was very high. It may not be possible, and may even be risky, to eliminate functions from operating systems or environments. Generally, one eliminates the functionality by preventing certain modules from loading; however, there may be interactions with other software modules in the systems that are not obvious to the user. This interdependency, particularly between apparently unrelated system modules, may cause the software to unpredictably execute or to halt. If the NDI developer designed the product, these interdependencies will be minimal.

D.6.7 Run-Time Versions

Some operating systems and environments have different development and run-time versions. The run-time versions already have unnecessary functionality removed. This allows the use of the full version on development workstations. Execution and testing use only those functions available in the run-time version, allowing the developers more flexibility during design. Some of these systems, such as VxWorks®, are rapidly becoming the OS of choice for command and control, fire control, and weapon control systems. System developers should consider the availability of run-time versions when selecting an operating system or environment.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

D.6.8 Watchdog Timers

The purpose of a watchdog timer is to prevent processors from entering loops that go on indefinitely or to exit processing that takes longer than expected. Applications software can use similar timers for safety-critical timing constraints. Watchdog timers issue an interrupt to the processor after a pre-determined time. A command from the applications software resets the timer to its pre-determined value each cycle. Software designers must not embed the reset command within a loop other than the main execution loop in the program.

In the design and implementation of the watchdog timer, the safety engineer addresses several issues. The watchdog timer processing should return safety-critical outputs and external system components to a safe state. Often, when a processor enters an infinite loop, the processor state and hence the system state is non-deterministic. Therefore, the safety engineer has no assurance that the system is in a safe state. The watchdog timer must be independent of the processing (i.e., not an imbedded function within the processor software unless that processing is completely independent of the timing loop it is monitoring).

The safety engineer should determine those processes that may adversely affect the safety of the system should the process execution time go beyond a pre-determined value. An example of such a process is a call to another subroutine that obtains a safety-critical value from an external source. The external source data is not available and the subroutine waits for the external source to provide it. If a delay in this data causes a data senescence problem in the safety-critical process, the program should interrupt the subroutine waiting on the data and return to a safe state. That safe state may be to refresh stale data or it may require the software to actively change the state of external system components. The watchdog timer may perform this function for a small system, or the design may implement secondary watchdog timers specifically for this type of function.

D.6.9 Configuration Management

Once the system developer determines the NDI software for the system, the developer should attempt to maintain configuration control over that software just as they do over the applications software. For commercially obtained items, this may require an agreement between the vendor and the developer. Even if an NDI supplier is unwilling to provide notification of changes for the product, the system developer can establish procedures to detect changes and determine potential impact. In the simplest case, detection may use file comparison programs to compare two or more copies of a product. However, these generally only detect a difference without providing any indication of the changes made. Refer to Section C.10 for further guidance on software configuration control.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

D.6.10 Prototyping

Although prototyping is not directly related to the application of NDI software in safety-critical systems, some of the benefits derived from this development methodology apply to the safety assessment of the application software in the NDI environment. Rapid prototyping allows the safety analyst to participate in the "build-a-little, test a little" process by analyzing and testing relatively small portions of system functionality each time. The safety analyst can then identify safety issues and incrementally incorporate them into the design. The safety analyst also begins to build an analytical picture of the interactions between the NDI software and the applications software. This allows the development and integration of risk mitigation techniques on a real-time basis.

D.6.11 Testing

Testing at the functional and system levels helps evaluate the risk associated with NDI in safety-critical systems. However, there are numerous limitations associated with testing. It is impossible to examine all possible paths, conditions, timing, and data senescence problems; create all possible failure conditions and modes; and cause the system to enter every state machine configuration that may occur in the operational environment. Other limitations to testing for safety that apply to any developmental system apply to the NDI environment as well. Earlier paragraphs in this Appendix addressed other limitations associated with testing. As with safety testing in general, the analyst can develop directed tests that focus on the safety-critical aspects of the interaction of the NDI software with the applications software. However, the ability to introduce failure modes is limited in this environment. Testers must have a set of benchmark (regression) tests that evaluate the full spectrum of the application software's safety-critical functionality when evaluating changes to the NDI software.

D.7 Safety Case Development for COTS/NDI Software

The final step is to build a safety case for (or against) the COTS item. This step involves assessing the overall safety mishap risk of the COTS item and providing recommended action to accept, reject, further evaluate (test or analysis), or modify. This decision is based on several factors, including:

- The COTS item functional criticality rating—safety critical, safety related, or not safety significant
- The amount and quality of the design and test data available
- The COTS contribution to system mishaps, hazards, failure modes, and causal factors
- How well the COTS item satisfies design SSRs and mitigates known safety risks
- How well the COTS item satisfies integration and qualification test requirements.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The amount of detailed safety analysis and testing of the COTS item is based on its functional criticality rating. For safety-critical items, more stringent testing and analysis are necessary to provide confidence in the safety of the system with the COTS item integrated into the system.

D.8 Summary

The techniques discussed in this appendix will not reduce the residual mishap risk associated with systems employing COTS unless they are part of a comprehensive SSP. A single technique may not be adequate for highly critical applications, and the system developer may have to use several approaches to reduce the risk to an acceptable level. Early identification of hazards and the development and incorporation of safety design requirements into the system design are essential elements to a cost-effective risk reduction process. The analysis of implementation in the design ensures that the software design meets the intent of the requirements provided. Early identification of the requirements for isolation software provides a cost-effective approach for addressing many of the potential safety issues associated with COTS and NDI applications.

Unfortunately, there are no silver bullets to resolve the safety issues with safety-critical applications of COTS, just as there are none for the software safety process in general. The techniques all require time and effort and involve some level of programmatic risk. This Handbook discusses the various techniques for reducing the risk of COTS applications in the context of safety-critical functions in the applications software. These same techniques are equally applicable to other desirable characteristics of the system, such as mission effectiveness, maintainability, and testability.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

APPENDIX E GENERIC SOFTWARE SAFETY REQUIREMENTS AND GUIDELINES

E.1 Introduction

The goal of this Appendix is to provide a set of generic safety requirements and guidelines for the design, development, and test of computing systems, software, and firmware that have or potentially have safety-critical or safety-related applications. If properly implemented, these requirements and guidelines will reduce the risk of the computing system causing an unsafe condition, the malfunction of a fail-safe system, or non-operation of a safety function. The requirements and guidelines may be used as a checklist to assess completeness or coverage of safety analyses and tests performed. New practitioners are warned that checklists are not sufficient in and of themselves. Just as applicable standards and directives must be tailored to the system or system type under development, so too must this set of requirements and guidelines be tailored. At the same time, these safety requirements and guidelines must be used concurrently with and within accepted systems engineering and software engineering practices, including configuration control, reviews and audits, structured design, and related systems engineering practices.

E.1.1 Determination of Safety-Critical Computing System Functions

The guidelines of this Appendix are to be used in determining which computing system functions are safety critical. Identification of safety-critical computing systems should be performed using safety assessment requirements or similar techniques. Software and firmware computing systems should be addressed as part of the safety assessment.

E.1.1.1 Specifications

The required safety functions of the computing system are determined from the analysis of the system and specifications. These computing system safety functions are to be designated safety-critical computing system functions (SCCSFs).

Identification of SCCSFs allows the safety program to focus efforts on those software functions that initiate, control, or monitor hazardous hardware or operations that could contribute to or provide causal influence to a mishap (accident) if not safely implemented.

E.1.2 Safety Criticality

“Safety critical” is defined in MIL-STD-882D as a term applied to any condition, event, operation, process, or item whose proper recognition, control, performance, or tolerance is

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

essential to safe system operation and support (e.g., safety-critical function, safety-critical path, or safety-critical component).

Hardware, human action, or software domains may include safety-critical functions. A safety-critical function at the system level will usually include all three components (hardware, human, and software) working together to perform a function that controls an energy-producing event or critical operation.

The emphasis of a software safety program is to identify those software or firmware components that support the system-level SCF designated as SCCSFs. Once identified, SCCSFs are classified by criticality (hazardous event control (severity potential) and software control over the hazardous event (autonomy)). Classification of SCCSFs based on level of control defines the levels of analysis and test required to ensure proper software design and implementation.

E.1.3 Identification of Safety-Critical Computing System Functions

SCCSFs can be categorized as safety application functions or safety infrastructure functions (SIFs). Safety application functions provide the logic within an application to directly control hazardous hardware and events. SIFs provide the support functions or services necessary to facilitate execution of the application logic and processing threads.

E.1.3.1 Safety Application Functions

Safety application functions are implemented functional application threads in which a failure, fault, or flaw in the software design or implementation could lead directly to a mishap. Safety application functions maintain the direct interface and influence over hazardous hardware or events. Loss of control over hazardous hardware, failure to monitor, or failure to provide mitigation for detected hazardous conditions make safety application functions prime candidates for a high degree of rigorous test and analysis to ensure proper functionality. Safety application functions can be categorized as safety critical or safety related based on software control category. Safety application function subcategories include:

- Safety-Critical Control Functions – Software functions that directly control the nominal pathway to hazardous hardware or events (e.g., arm commands, launch and firing commands, track identification determination, and ship turning and speed commands).
- Safety-Critical Monitor Functions – Software functions that monitor the nominal control pathway for abnormalities or monitor hazardous hardware conditions (e.g., temperature, speed, acceleration, position, mode, and weapon system status).
- Safety-Critical Hazard Mitigation Functions – Software functions that act on the monitor function to directly mitigate a hazardous condition (e.g., auto break engage, safe booster, abort launch, and shutdown equipment) or provide indications that a hazardous condition exists (e.g., visual alerts and audible alarms).

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Safety-Critical Display Functions – Software functions that provide displays for hazardous hardware or events that must be acted upon to mitigate a potential mishap (e.g., friendly about to be engaged and weapon armed but should be safe). The display of incorrect information or failure to provide the correct display within a specified time can lead to a mishap.

E.1.3.2 Safety Infrastructure Functions

SIFs are implemented support threads and services in which a failure, fault, or flaw in the software design or implementation could provide the opportunity for a mishap. SIFs provide support to the overall function of the software system and include program loading, system state or mode control, and system health monitoring. Safety application functions can be categorized as safety critical or safety related based on software control category; however, because SIFs support a safety-critical function, they are generally designated as safety critical. SIF subcategories include:

- Safety-Critical Program Load Functions – Software functions that provide system start-up, monitoring, and built-in-test events and processing designed to put the program and its interfaces in a safe state upon initialization.
- Safety-Critical Operational Environment Functions – Software functions that provide system resource, database, timing, interrupt, and general operating system functionality to support client services.
- Safety-Critical Stable State Functions – Software functions that provide the necessary system logic to remain or transition to a defined operational (e.g., tactical, training, and test), degraded operational (e.g., loss of data center, loss of node), or logistic configuration (e.g., on, off, and maintenance).
- Safety-Critical Fault Monitoring Functions – The events and processes associated with monitoring the safety-critical system health, including error processing, memory protection, and protection from uncertified interfaces.
- Safety-Critical Data Transport Functions – Software functions that provide packing and unpacking services of safety-critical data, objects, or messages prior to and after transmission over an interface. This interface may be over copper path, fiber optics, or air data link.
- Safety-Critical Shutdown Functions – The events and processes necessary to return the system to a known, uncorrupted state during process termination or system shutdown.

E.1.3.3 Common SCCSFs

When identifying SCCSFs, all levels of the computing system must be considered, including the operating environment, operating system, application software, computing hardware, and firmware components.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Any software or firmware function that controls or directly influences the pre-arming, arming, enabling, release, launch, firing, or detonation of a weapon system, including target identification, selection, and designation
- Any software or firmware function that determines, controls, or directly influences the flight path of a weapon system
- Any software or firmware function that controls or directly influences the movement of gun mounts, launchers, and other equipment, including the pointing and firing safety of that equipment
- Any software or firmware function that controls or directly influences the movement of munitions or hazardous materials
- Any software or firmware function that monitors the state of the system for ensuring its safety
- Any software or firmware function that senses hazards or displays information concerning the protection of the system
- Any software or firmware function that controls or regulates energy sources in the system
- Fault detection priority; the priority structure for fault detection, restoration of safety, or correcting logic for safety-critical processes. Software units or modules handle or respond to these faults.
- Interrupt processing software, interrupt priority schemes, and routines that disable or enable interrupts
- Software or firmware components that have autonomous control over safety-critical hardware
- Software or firmware which generates signals that directly influence or control the movement of potentially hazardous hardware components or initiate safety-critical actions
- Software or firmware that generates outputs that display the status of safety-critical hardware systems. Where possible, these outputs should be duplicated by non-software and firmware generated output.
- Software or firmware used to compute safety-critical data, including applications software and firmware that may not be connected to or directly control a safety-critical hardware system (e.g., stress analysis programs).

E.1.4 Requirement Types

E.1.4.1 Behavioral

A behavioral requirement describes user interface issues, system usage, and how a system fulfills a specific function. For a requirement to exhibit “behaviors,” there must be an act based on an input stimulus. This act is typically recognizable as an “Input > Process > Output thread” where the input is a stimulus, the process is the act, and the output is the product of the act. Behavioral requirements are also known as functional requirements.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

Behavioral safety requirements are those requirements that are part of an SCCSF that flow down to the software requirements specifications. A behavioral requirement may have duplicate meanings where a single requirement can have both safety and performance characteristics. A pure performance requirement cannot have any safety characteristics. The system safety analyst must determine if the behaviors of a single requirement or series of requirements (function) could subject people, property, or the environment to risk if incorrectly defined or implemented. If so, that requirement or series of requirements has safety relevance.

E.1.4.2 Non-Behavioral

A non-behavioral (non-functional) requirement describes a technical feature(s) of a system, usually pertaining to:

- Availability (24 hours/day, 7 days/week)
- Security (only cleared personnel are authorized to log in to the system)
- Performance (real time vs. non-real time)
- Interoperability (able to communicate in a C4I net).

An example of a non-behavioral requirement would be "The final operational program is not permitted to have dead code." Dead code does not have behaviors; however, the existence of dead code could cause a hazard if inadvertently executed or injected during software maintenance activities. Non-behavioral requirements may or may not flow down to the SRS. This decision is made based on the level of abstraction validation for the requirement (component, element, or system).

E.1.4.3 Design Constraint

Design constraint requirements are absolute requirements in the form of constraints placed upon a system design. There are many different designs that can satisfy a specific requirement. The intent of a design constraint is to limit the design to a specific method. Examples of design constraints in computing systems and design include:

- Computing language
- Communication method
- Use of common design patterns
- Use of specific computing environment
- Timing constraints
- Priority schemas.

Design constraint requirements may or may not flow down to the SRS. This decision is made based on the level of abstraction validation for the requirement (component, element, or system).

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.1.5 Implementation of Generic Requirements and Guidelines

The purpose of generic requirements and guidelines into design safety into the system. This section provides insight on how Sections E.2 through E.13 should be implemented as part of the system software engineering process. Keep in mind that not all generic requirements and guidelines described in the remainder of this Appendix may apply to the system under development because systems vary in size, shape, use, and complexity. Two implementation methods are presented. The first integrates the definition and flow-down of requirements during the system software engineering development process. The second allows the use of guidelines and requirements as a compliance assessment tool once development is complete.

E.1.5.1 Integration with the Software Engineering Process

The best implementation of defining safety requirements is in phase with the software development process, from initial definition of system requirements through final acceptance testing of the fielded system. Defining safety requirements in top-level specifications allows the natural decomposition to lower-level specifications and requirements. This process ensures flow-down from top-level requirements and facilitates traceability of safety requirements throughout the specification tree.

E.1.5.2 Audit Tool Post-Software Design or Development

If the system under analysis did not have safety designed into the system during phase engineering with the software development process, the requirements and guidelines may be used as an audit tool. If this method is chosen, a compliance matrix is developed and audits are conducted to determine the level of compliance with the generic requirements and guidelines.

E.1.5.3 Full Compliance vs. Partial or Non-Compliance

Use of either method described above requires a determination of compliance with the individual requirements or guidelines. This compliance can be validated through analysis, inspection, demonstration, or test. If full compliance with the specified requirement or guideline is met, no further action is required. If the requirement or guideline is determined to be either partially compliant or non-compliant, it is the responsibility of the safety analyst to conduct a safety risk assessment. The safety risk assessment must quantify any safety risk incurred based on the partial or non-compliance results documented in the hazard tracking system.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.2 Design and Development Process Requirements and Guidelines

The requirements and guidelines in this section apply to the design and development phases. The requirements are indicated by explicit "shall" statements. The use of "should" and "will" in other statements is provided as guidance.

E.2.1 Configuration Control

Configuration control shall be established as soon as practical in the system development process. The software Configuration Control Board must approve all software changes prior to their implementation after an initial baseline has been established. A member of the CCB with safety experience or a consistent capability to recognize safety issues shall be tasked with the responsibility for evaluating all software changes for potential safety impact. This CCB member should be a member of the system safety engineering team. A member of the hardware CCB shall be a member of the software CCB and vice versa to keep members apprised of hardware changes and to ensure that software changes do not conflict with or introduce potential safety hazards due to hardware incompatibilities. There shall be a specific attribute in the software change documentation (e.g., Software Trouble Report) that denotes if the potential change (e.g., requirement, architecture, or code) impacts safety. This attribute can take the form of "Safety = Yes or No."

E.2.2 Software Quality Assurance Program

A Software Quality Assurance Program shall be established for systems having safety-critical functions. SQA assurance is the confidence, based on objective evidence, that the risk associated with using a system conforms to expectations of or willingness to tolerate risk. Several issues should be addressed by the program office to calibrate confidence in the software. There is consensus in the software development community that no one assurance approach is adequate for critical software assurance and that some integration of the evidence provided by these various approaches must be used to make confidence decisions.

E.2.3 Two Person Rule

At least two people shall be thoroughly familiar with the design, code, testing, and operation of each software module in the system.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.2.4 Program Patch and Overlay Prohibition

Patches and overlays shall be prohibited throughout the development process. All software changes shall be coded in the source language and compiled prior to entry into operational or test equipment.

E.2.5 Software Design Verification and Validation

The software shall be analyzed throughout the design, development, and maintenance processes by a system safety engineering team to verify and validate that the safety design requirements have been correctly and completely implemented. Test results shall be analyzed to identify potential safety anomalies.

E.2.5.1 Correlation of Artifacts Analyzed to Artifacts Deployed

Much of the confidence placed in a critical software system is based on the results of tests and analyses performed on specific artifacts produced during system development (e.g., source code modules and executable programs produced from the source code). The results of these tests and analyses contribute to confidence in the deployed system only to the extent that the tested and analyzed components are actually in the deployed system. There have been cases where the wrong version of a component has accidentally been introduced into a deployed system and caused unexpected failures or presented a potential hazard.

- Does the SDP describe a thorough CM process that includes version identification, access control, change audits, and the ability to restore previous revisions of the system?
- Does the CM process rely entirely on manual compliance, or is it supported and enforced by tools?
- Does the CM process include the ability to audit the version of specific components (e.g., through the introduction of version identifiers in the source code that are carried through into the executable object code)? If not, how is process enforcement audited (e.g., for a given executable image, how can the versions of the components be determined)?
- Is there evidence in the design and source code that the CM process is being adhered to (e.g., are version identifiers present in the source code if this is part of the CM process described)?
- During formal testing, are there any problems with inconsistent or unexpected versions?

Tool integrity is the second issue that impacts confidence of correlation between the artifacts analyzed and those actually deployed. Software tools (e.g., computer programs used to analyze, transform, or otherwise measure or manipulate products of a software development effort) can

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

have an impact on the level of confidence placed in critical software. For example, all analysis of the source code can be undermined if the compiler used on the project is buggy. In situations where this is a potential issue (e.g., the certification of digital avionics), a distinction is drawn between two classes of tools:

- Those that transform the programs or data used in the operational system and can therefore actively introduce unexpected behavior into the system
- Those used to evaluate the system and therefore can contribute (at worst) to not detecting a defect.

There is a limit to how many resources should be applied. For example, in validating the correctness of an Ada compiler, a developer may reasonably argue that until there is evidence of a problem, it is sufficient mitigation to use widely-used commercially available tools. The program office should have confidence that the developer is addressing these issues. Do the SDP and risk reduction plan include a description of tool qualification criteria and plans? Does the plan include a description of what the critical tools are (e.g., compiler and linker loader) and what the risk mitigation approach is (e.g., use widely available commercial compilers, establish a good support relationship with the vendor, and canvas other users of the tools for any known problems)? Is there any evidence in the design documentation or source code of work-arounds being introduced to accommodate issues encountered in critical tools? If so, what steps are being taken to ensure that the problems are fixed and that these issues do not result in a reduced confidence in the tools?

E.2.5.2 Correlation of Process Reviewed to Process Employed

Process integrity is the next issue that impacts confidence of correlation between the artifacts analyzed and those actually deployed. Processes are designed to implement a particular systems engineering task and standardize the internal task steps so that output is predictably similar.

- Confidence in the process used to develop critical software is a key part of overall confidence in the final system. However, that confidence is justified only if there is reason to believe that the process described is the process applied. The program office can use milestone reviews as a way to audit process enforcement and adherence to the processes described. The use of static analysis and inspection of artifacts (e.g., design documentation, source code, and test plans) can provide increased confidence that the process is being adhered to (or expose violations of the described process, which should be given immediate attention).
- Are the processes described in the SDP enforceable and auditable? Specific coding standards or testing strategies can be enforced and independently audited by a review of the products. Vague or incomplete process descriptions can be difficult to enforce and determine if they are being adhered to, which reduces the confidence they provide with respect to critical software assurance arguments.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- As the development progresses, what is the overall track record of compliance with the processes described in the SDP (as determined by compliance audits during milestone reviews)? If there is reason for concern, this should become a separate topic for resolution between the program office and the developer.
- How does the DA monitor and enforce process compliance by the subcontractors? Is there evidence that this is being accomplished?

E.2.5.3 Reviews and Audits

Desk audits, peer reviews, static and dynamic analysis tools and techniques, and debugging tools shall be used to verify implementation of design requirements in the source code. Desk audits by a single person were nullified by standards bodies, such as IEEE and ISO, by the year 2000 (see IEEE 1028) but can still be used for local low-level quality assurance credit and metrics. System safety can no longer justify these low-level assurances for credit in mitigating a hazard. Particular attention should be paid to the implementation of identified safety-critical computing system functions and the requirements and guidelines provided in this document by higher-level quality assurance methodologies. Reviews of the software source code shall ensure that the code and comments within the code agree.

E.3 System Design Requirements and Guidelines

The requirements and guidelines of this section apply to the general system design.

E.3.1 Designed Safe States

The system shall have at least one safe state identified for each logistic and operational phase. Safe states shall be fail-safe such that mishaps are avoided in the event that the software does not execute or executes improperly.

E.3.2 Safe State Return

The software shall return hardware subsystems under control of software to a designed safe state when unsafe conditions are detected. Conditions that can be safely overridden by the battle short shall be identified and analyses performed to verify their safe incorporation.

E.3.3 Stand-Alone Computer

Where practical, safety-critical functions should be performed on a stand-alone computer. If this is not practical, safety-critical functions shall be isolated, to the maximum extent practical, from non-critical functions.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.3.4 Ease of Maintenance

The system and software shall be designed for ease of maintenance by personnel that are not associated with the original design team. Documentation specified for the computing system shall be developed to facilitate maintenance of the software. Strict configuration control of the software during development and after deployment is required. It is recommended that techniques for the decomposition of the software system be used for ease of maintenance.

E.3.5 Restoration of Interlocks

Upon completion of tests and training where safety interlocks are removed, disabled, or bypassed, restoration of those interlocks shall be verified by the software prior to resuming normal operation. While overridden, a display of the status of the interlocks shall be made on the operator's or test conductor's console, if applicable.

E.3.6 Input and Output Registers

Input and output registers and ports shall not be used for both safety-critical and non-critical functions unless the same safety design criteria are applied to the non-critical functions.

E.3.7 External Hardware Failures

The software shall be designed to detect failures in external input or output hardware devices and revert to a safe state upon occurrence. The design shall consider potential failure modes of the hardware involved.

E.3.8 Safety Kernel Failure

The system shall be designed such that a failure of the safety kernel (when implemented) will be detected and the system returned to a designated safe state.

E.3.9 Circumvent Unsafe Conditions

The system design shall not permit detected unsafe conditions to be circumvented. If a battle short or safety arc condition is required in the system, it shall be designed such that it cannot be inadvertently activated or activated without authorization.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.3.10 Fallback and Recovery

The system shall be designed to include fallback and recovery to a designed safe state of reduced system functional capability in the event of a failure of system components.

E.3.11 Simulators

If simulated items, simulators, and test sets are required, the system shall be designed such that the identification of the devices is fail-safe and that operational hardware cannot be inadvertently identified as a simulated item, simulator, or test set.

E.3.12 System Errors Log

The software shall make provisions for logging all system errors. The operator shall have the capability to review logged system errors. Errors in safety-critical routines shall be highlighted and shall be brought to the operator's attention as soon after occurrence as practical.

E.3.13 Positive Feedback Mechanisms

Software control of critical functions shall have feedback mechanisms that give positive indications of the function's occurrence.

E.3.14 Peak Load Conditions

The system and software shall be designed to ensure that design safety requirements are not violated under peak load conditions.

E.3.15 Endurance Issues

Although software does not wear out, the context in which a program executes can degrade with time. Systems that are expected to operate continuously are subjected to demands for endurance—the ability to execute for the required period of time without failure. For example, the failure of a Patriot missile battery in Dhahran during the Persian Gulf War was traced to the continuous execution of tracking and guidance software for over 100 hours; the system was designed and tested against a 24-hour upper limit for continuous operation. Long-duration programs are exposed to a number of performance and reliability issues that are not always obvious and that are difficult to expose through testing. This makes a careful analysis of potential endurance-related defects an important risk-reduction activity for software to be used in continuous operation. Examples include:

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Has the developer explicitly identified the duration requirements for the system? Has the developer analyzed the behavior of the design and implementation if these duration assumptions are violated? Are any of these violations a potential hazard?
- Has the developer identified potential exposure to the exhaustion of finite resources over time, and are adequate detection and recovery mechanisms in place to handle these? Examples include:
  - Memory (e.g., heap leaks from incomplete software storage reclamation)
  - File handles and transmission control protocol ports (if not closed under error conditions)
  - Counter overflow (e.g., 8-bit counter and > 255 events were factors in the failure of Therac-25 radiation treatment machines).
- Has the developer identified potential exposure to performance degradation over time, and are adequate deduction and recovery mechanisms in place to handle these? Examples include memory and disk fragmentation that can result in increased latency.
- Has the developer analyzed increased exposure to cumulative effects over time, and are adequate detection and recovery mechanisms in place to handle these effects so that they do not present hazards? Examples include cumulative drift in clocks, cumulative jitter in scheduling operations, and cumulative rounding errors in floating point and fixed-point operations.

E.3.16 Error Handling

Causal analyses of software defects frequently identify error handling as a problem area. For example, one industry study observed that a common defect encountered was failure to consider all error conditions or error paths. A published case study of a fault-tolerant switching system indicated that approximately two-thirds of the system failures that were traceable to design faults were due to faults in the portion of the system that was responsible for detecting and responding to error conditions. The results of a Missile Test and Readiness Equipment internal research project on error handling in large software systems also indicate that error handling is a problematic issue for software systems. In many cases, the issues exposed were the result of oversight or logic errors. It is important to note that these types of errors have been encountered in software that is far along in the development process and under careful scrutiny because it is mission-critical software. The presence of simple logic errors illustrates that error handling is often not as carefully inspected and tested as other aspects of system design. It is important that the program office have adequate insight into the developer's treatment of error handling in critical systems. Consideration include:

- Has the developer clearly identified an overall policy for error handling? Have the specific error detection and recovery situations been adequately analyzed? Has the developer defined the relationship between exceptions, faults, and unexpected results?
- Are different mechanisms used to convey the status of computations? What are these mechanisms (e.g., Ada exceptions, OS signals, return codes, and messages)? If return

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

codes and exceptions are both used, are there guidelines for when each is to be used? What are these guidelines, the rationale for using them, and how are they enforced? Are return codes and exceptions used in distinct layers of abstraction (e.g., return codes only in calls to COTS OS services) or freely intermixed throughout the application? How are return codes and exceptions mapped to each other? In this mapping, what occurs if an unexpected return code is returned or an unexpected exception is encountered?

- Has the developer determined the costs of using exceptions for their compiler(s)? What is the space and runtime overhead of having one or more exception handlers in a sub-program and a block statement, and is the overhead fixed or a function of the number of handlers? How expensive is propagation, both explicit and implicit?
- Are derived types used? If so, are there any guidelines regarding the exceptions that can be raised by the operations associated with the derived types? How are they enforced?
- Are there guidelines regarding exceptions that can be propagated during task rendezvous? How are they reinforced and tested?
- Is program suppression ever used? If so, what are the restrictions on its use, and how are they enforced? What is the rationale for using or not using program suppression? If program suppression is used, are there guidelines for checking that must be in the code for critical constraints in lieu of the implicit constraint checks? If not, how is the reliability of the code ensured?
- Are there any restrictions on the use of tasks in declarative regions of sub-programs (i.e., sub-programs with dependent tasks)? If so, how are they enforced? How are dependent tasks terminated when the master sub-program is terminating with an exception, and how is the suspense of exception propagation until dependent task termination handled?
- What process enforcement mechanisms are used to ensure global consistency among error handling components (e.g., for systems where various subcontractors were constrained, they each made plausible design decisions regarding error handling policy, but when these components were integrated, they were discovered to be inconsistent)?
- Are there guidelines for when exceptions are masked (e.g., a handler for an exception does not propagate an exception), mapped (e.g., a handler for an exception propagates a different exception), or propagated? If so, how are they enforced? Are there any restrictions on the use of the other handlers? If so, how are they enforced?
- How does the developer ensure that return codes or status parameters are checked after every subroutine call or ensure that failure to check them does not present a hazard?
- Are there any restrictions on the use of exceptions during elaboration (e.g., checking data passed to a generic package during installation)? Is exception handling during elaboration a possibility due to initialization functions in declarative regions? If so, how is this handling tested, and are there design guidelines for exception handling during elaboration? If not, how are they assured that this does not present a hazard?

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.3.17 Redundancy Management

Redundancy is frequently employed to reduce the vulnerability of a software system to a single mechanical or logic failure. However, the added complexity of managing the redundancy in fault-tolerant systems may make them vulnerable to additional failure modes that must be accounted for by the developer. For example, the first shuttle flight and the 44th flight of NASA's Advanced Fighter Technology Integration (AFTI)-F16 software both exhibited issues associated with redundancy management. The first shuttle flight was stopped 20 minutes before scheduled launch because of a race condition between the two versions of the software. The AFTI-F16 had problems related to sensor skew and control law gain, causing the system to fail when each channel declared the others had failed. The analog backup was not selected because the simultaneous failure of two channels was not anticipated.

When the developer's design includes redundancy (e.g., duplicate independent hardware or N version programming), the additional potential failure modes invoked by the redundancy scheme itself must be identified and mitigated. Examples include sensor skew, multiple inconsistent states, and common mode failures.

E.3.18 Safe Modes and Recovery

A common design idiom for critical software systems is that they are "self checking and self protecting." This means that software components protect themselves from invalid requests or invalid input data by frequently checking for violations of assumptions or constraints. In addition, software components check the results of service requests to other system components to ensure that they are functioning as expected. Such systems typically provide for the checking of internal intermediate states to determine if the routine is working as expected. Violations of any of these checks can require transition to a safe state if the failure is serious or if the confidence in further correct execution has been reduced. Failure to address this defensive approach can result in a variety of failures propagating throughout the system in unexpected and unpredictable ways, potentially resulting in a hazard. The following questions provide a baseline for the analysis.

- Does the developer identify a distinct safe mode or set of safe modes? Has the analysis of these safe modes adequately considered the transition to these safe modes from potentially hazardous states (e.g., internal inconsistency)?
- Does the design include acceptable safety provisions when an unsafe state is detected?
- Does the design include assertion checks or other mechanisms for the regular run-time calibration of internal logic consistency?
- Does the developer provide for an orderly system shutdown as a result of operator shutdown instructions, power failure, etc.?
- Does the developer explicitly define the protocols for any interactions between the system and the operational environment? If anything other than the expected

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

sequences or interlocks is encountered, does the system design detect this and transition to a safe state?

- Does the developer account for all power up, self test, and handshaking with other components in the operational environment to ensure execution begins in a predicted and safe state?

E.3.19 Isolation and Modularity

The practical limits on resources for critical software assurance are consistent with the consensus in the software development community that a major design goal for critical software is to keep the critical portions small and isolated from the rest of the system. The program office evaluates evidence provided by the developer that indicates the extent to which this isolation has been a design goal and the extent to which the implementation has successfully realized this goal. Confidence that unanticipated events or latent defects in the rest of the software will not introduce an operational hazard correlates with the confidence that such isolation has been achieved. Example considerations are:

- Does the developer's design provide explicit evidence of an analysis of the criticality of the components and functions (i.e., does the design reflect an analysis of which functions can introduce a hazard)?
- Does the developer's design and implementation provide evidence that coupling has been kept to a minimum in critical portions of the software (e.g., restrictions on shared variables and side-effects for procedures and functions)?
- Does the developer's design include the implementation of firewalls in the software? Do critical portions of code perform consistency checking of data values provided by clients (software using the critical software as a service) and the software services the critical software calls (e.g., OS services)?
- Does the critical software design and implementation include explicit checks of intermediate states during computation to detect possible corruption of the computing environment (e.g., range checking for an intermediate product in an algorithm)?
- Does the developer provide the criteria for determining what software is critical, and is there evidence that these criteria were applied to the entire software system? How does the developer provide evidence that the portions considered non-critical will not introduce a hazard?

E.4 Power-Up System Initialization Requirements

The following requirements apply to the design of the power subsystem, power control, and power-on initialization for safety-critical applications of computing systems.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.4.1 Power-Up Initialization

The system shall be designed to power up in a safe state. An initialization test shall be incorporated in the design that verifies that the system is in a safe state and that safety-critical circuits and components are tested to ensure their safe operation. The test shall also verify memory integrity and program load.

E.4.2 Power Faults

The system and computing system shall be designed to ensure that the system is in a safe state during power up, intermittent faults, or fluctuations in power that could adversely affect the system. The system and software shall be designed to provide for a safe, orderly shutdown of the system during a fault or power down such that potentially unsafe states are not created.

E.4.3 Primary Computer Failure

The system shall be designed such that a failure of the primary control computer will be detected and the system returned to a safe state.

E.4.4 Maintenance Interlocks

Maintenance interlocks, safety interlocks, safety handles, or safety pins shall be provided to preclude hazards to personnel maintaining the computing system and associated equipment. Where these interlocks must be overridden to perform tests or maintenance, they shall be designed so they cannot be inadvertently overridden or left in the overridden state once the system is restored to operational use. The override of interlocks shall not be controlled by a computing system.

E.4.5 System-Level Check

The software shall be designed to perform a system-level check at power up to verify that the system is safe and functioning properly prior to application of power to safety-critical functions, including hardware controlled by the software. Periodic tests shall be performed by the software to monitor the safe state of the system.

E.4.6 Control Flow Defects

Control flow refers to the sequencing of instructions executed while a program is running. The consequences of defects in control flow may include program termination (e.g., an Ada exception propagates to the outermost scope, or the program attempts to execute an illegal instruction to an invalid region of memory). It can be difficult to detect the consequence(s) of a

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

control flow defect that may continue execution in an invalid or unpredictable state. For example, a "computed go-to" (e.g., using base and displacement registers in an assembly language program) may branch to a legitimate instruction sequence that is not the correct sequence given the current state of the system. Therefore, for critical systems, evidence must be presented that these kinds of defects are avoided or mitigated through the evaluation of items such as those below.

- If the developer is using assembly language, are there any computed control-flow statements? Are there any branches or jumps to an address that is computed (e.g., base and displacement registers) rather than a static symbolic label? If so, how does the developer ensure that these address computations never result in a "wild jump" or that such wild jumps do not represent a hazard?
- In Ada functions, there may be paths where control can "fall through" (i.e., the function terminates in a statement other than a return or an exception propagation). This is an invalid control flow and will result in the propagation of the pre-defined Ada exception Program_Error. How does the developer ensure that this will not happen or that the propagation of Program_Error from a function will not represent a hazard?
- If the developer is using the C programming language, is the C facility of passing addresses of functions as arguments ("funargs") used? If so, how does the developer ensure that all calls to a function pointed to by a funarg are valid and that no hazards result from invalid funargs?
- If Ada is used, how does the developer ensure that no exception can propagate to the outermost scope and terminate the program, or how is this dealt with so that such termination is not a hazard? Are restrictions on exceptions that can be propagated from specific routines present (e.g., are only a restricted set of exceptions allowed to propagate to a caller)? If there are such restrictions, how are they enforced? If not, how does the developer provide assurance that all potential exceptions are handled?
- A second timing-related failure mode for software is the existence of race conditions. Race conditions are activities that execute concurrently and for which the result depends on the sequencing of activity. For example, if Ada tasking is used and two tasks concurrently access and change a device, the final result of the computations may depend on which task is the first to access the device (the task that "won the race"). In Ada, the scheduling of eligible concurrent tasks is non-deterministic, which means that two consecutive executions with the same data may run differently. Note that the race need not be between tasks; in the fatal software failures of the Therac-25 radiation treatment devices, one failure mode was a race condition between internal tasks and the operator's timing for screen update and entry. These failure modes are often difficult to isolate, repeat, and correct once a system has been deployed; they are equally difficult to detect during testing and have caused subtle latent defects in deployed software (e.g., the first attempt to launch the Columbia Space Shuttle was aborted 20 minutes before launch due to a latent race condition that had a 1-in-67 chance of being exposed at each system powerup). The program office should look for evidence that all potential hazards resulting from timing and

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

sequencing have been systematically considered and any hazards that are identified are mitigated.

- Has the developer clearly presented the concurrent requirements (explicit or derived) for the system? Have the timing and sequencing consequences been given significant attention with respect to repeatable behavior and hazard identification and mitigation for the concurrent requirements?
- Has the developer identified all real-time requirements (e.g., reading sensor data, interacting with devices, and constraints imposed by other systems)? Would the consequences of failing to meet any of those requirements represent a hazard? If so, what hazard mitigation has the developer used? Note that real-time requirements include deadlines and upper and lower bounds on event timing (e.g., the minimum interval between consecutive packets on a communications channel or time-out triggers).
- If there are any real-time requirements that are critical (i.e., failing to meet them would present a hazard), how has the developer identified and controlled all sources of unpredictable timing, including upper and lower bounds on device latency (e.g., secondary storage), upper and lower bounds on the timing characteristics of Ada language features that may widely vary (e.g., exception propagation, dynamic memory for access types, delay statement, task rendezvous, and termination), and upper and lower bounds on the software's interaction with other subsystems (e.g., burst mode or failed communications, data rates exceeding or below expected values, and time-outs for failed hardware).
- Where there is potential interference or shared data among multiple threads of control (e.g., Ada tasks and OS processes) or multiple interrupt handlers, have all control and data flows been identified by the developer? If so, has the developer identified all potential race conditions? How has the developer ensured that there are no race conditions that present a hazard or that such hazards are mitigated? Note that in Ada, the interleaved update of shared variables by multiple Ada tasks is erroneous; therefore, the results are unpredictable.
- Has the developer identified potential hazards resulting from sequencing errors? Even for single threads of control, there are potential failure modes related to sequencing errors. For example, in Ada, calling a function before the function body has been elaborated is an example of access before elaboration and results in a Program_Error being raised. The initial elaboration sequence is an important aspect of program correctness for non-trivial Ada programs. This is also an example of potential sequencing failures that the developer should review for identification of possible hazards and mitigate any such hazards discovered. Another sequencing failure example is calling an operation before performing any required initialization.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.5 Computing System Environment Requirements and Guidelines

The requirements and guidelines of this section apply to the design and selection of computers, microprocessors, programming languages, and memory for safety-critical applications in computing systems.

E.5.1 Hardware and Hardware/Software Interface Requirements

- CPU
- Memory
- Failure in the computing environment
- Hardware and software interfaces
- Self-test features
- Watchdog timers, periodic memory checks, and operational checks
- System utilities
- Compilers, assemblers, translators, and operating systems
- Diagnostic and maintenance features
- Memory diagnosis.

E.5.1.1 Failure in the Computing Environment

An application program exists in the context of a computing environment—the software and hardware that collectively support the execution of the program. Failures in this environment can result in a variety of failures or unexpected behavior in the application program and must be considered in a hazard analysis. For some of these failure modes (e.g., program overwrite of storage), it is difficult to completely predict the consequences (often because it is dependent on the region that is overwritten and the pattern). The burden of proof is on the developer to provide evidence that there is no exposure to these types of failures or that such failures do not represent a potential hazard. The following identifies considerations in this regard.

- Has the developer identified the situations in which the application can corrupt the underlying computing environment? An example is the erroneous writing of data to the incorrect location in storage (e.g., writing to the 11th element of a 10-element array through pointer manipulation in C, or through unchecked conversion or use of the pragma Interface in Ada). Has Ada's pragma Suppress been used? If so, how does the developer ensure that such storage corruption is not missed by removing the runtime checks? Note that if pragma Suppress is used and the detection of a constraint violation is masked, the results are unpredictable (the program is erroneous). Has the developer provided evidence that the software's interaction with the hardware does not corrupt the computing environment in a way that introduces a

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

hazard (e.g., setting a program status word to an invalid state or sending invalid control sequences to a device controller)?

- Has the developer analyzed potential failure modes of the Ada Runtime Environment (ARTE), the host OS or executive, and any other software components (e.g., database management system) used in conjunction with the application for any hazards that they might introduce? What evidence does the developer provide that there are no failure modes that present a hazard or that the identified hazards have been mitigated? What evidence does the developer provide for the required level of confidence in the ARTE, OS, etc. (e.g., for commercial avionics certification and other safety-critical domains, high assurance or certified subset ARTEs have been used)?
- Has the developer provided evidence that data consistency management has been adequately addressed where it can impact critical functions? For example, is file system integrity checked at startup? Are file system transactions atomic, or is there a mechanism for backing out from corrupted transactions?

E.5.2 CPU Selection

The following guidelines apply to the selection of CPUs:

- CPUs that process entire instructions or data words are preferred to those that multiplex instructions or data (e.g., an 8-bit CPU is preferable to a 4-bit CPU emulating an 8-bit machine)
- CPUs with separate instructions, data memories, and buses are preferred to those using a common data/instruction bus. Alternately, memory protection hardware separating program memory and data memory, either segment or page protection, is acceptable.
- CPU flags are single point failures for comparison (IF) statements and counters
- CPUs, microprocessors, and computers that can be fully mathematically represented are preferable to those that cannot.

E.5.3 Minimum Clock Cycles

For CPUs that do not comply with the guidelines above or those used at the limits of their design criteria (e.g., at or above maximum clock frequency), analyses and measurements shall be conducted to determine the minimum number of clock cycles that must occur between functions on the bus to ensure that invalid information is not picked up by the CPU. Analyses shall also be performed to ensure that interfacing devices are capable of providing valid data within the required time frame for CPU access.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.5.4 Read Only Memory

Where read only memory (ROM) is used, positive measures shall be taken to ensure that the data cannot be corrupted or destroyed.

E.6 Self-Check Design Requirements and Guidelines

The design requirements of this section provide for self-checking of the programs and computing system execution.

E.6.1 Watchdog Timers

Watchdog timers or similar devices shall be provided to ensure that the microprocessor or computer is operating properly. The timer reset shall be designed such that the software cannot enter an inner loop and reset the timer as part of that loop sequence. The design of the timer shall ensure that failure of the primary CPU clock cannot compromise its function. The timer reset shall be designed such that the system is returned to a known safe state and the operator is alerted (as applicable).

E.6.2 Memory Checks

Periodic checks of memory, instruction, and data bus(es) shall be performed. The design of the test sequence shall ensure that single point and multiple failures are detected. Checksum of data transfers and program load verification checks shall be performed at load time and periodically thereafter to ensure the integrity of safety-critical code.

E.6.3 Fault Detection

Fault detection and isolation programs shall be written for safety-critical subsystems of the computing system. The fault detection program shall be designed to detect potential safety-critical failures prior to the execution of the related safety-critical function. Fault isolation programs shall be designed to isolate the fault to the lowest level practical and provide this information to the operator or maintainer. In the Java language, failure to write for a thrown exception will result in lock-up of safety critical functions not related to the statement causing the exception.

E.6.4 Operational Checks

Operational checks of testable safety-critical system elements shall be made immediately prior to performance of a related safety-critical operation.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.7 Safety-Critical Computing System Functions Protection Requirements and Guidelines

The design requirements and guidelines of this section provide for the protection of safety-critical computing system functions and data.

E.7.1 Safety Degradation

Other interfacing systems, subsystems, software in the internal architecture, and clocks and timers shall be designed to prevent degradation of safety functions and capabilities, as well as timing of these functions and capabilities.

E.7.2 Unauthorized Interaction

The software shall be designed to prevent unauthorized system or subsystem interaction from initiating or sustaining a safety-critical function sequence.

E.7.3 Unauthorized Access

The system design shall prevent unauthorized or inadvertent access to or modification of the software (source or assembly) and object code. This includes preventing self-modification of the code.

E.7.4 Safety Kernel ROM

Safety kernels should reside in non-volatile ROM or in protected memory that cannot be overwritten by the computing system.

E.7.5 Safety Kernel Independence

A safety kernel, if implemented, shall be designed and implemented in a manner that cannot be corrupted, misdirected, delayed, or inhibited by any other program in the system.

E.7.6 Inadvertent Jumps

The system shall detect inadvertent jumps within or into SCCSFs; return the system to a safe state; and, if practical, perform diagnostics and fault isolation to determine the cause of the inadvertent jump.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.7.7 Load Data Integrity

The executive program or OS shall ensure the integrity of data or programs loaded into memory prior to their execution.

E.7.8 Operational Reconfiguration Integrity

The executive program or OS shall ensure the integrity of the data and programs during operational reconfiguration.

E.8 Interface Design Requirements

The design requirements of this section apply to the design of input/output interfaces.

E.8.1 Feedback Loops

Feedback loops from the system hardware shall be designed such that the software cannot cause a runaway condition due to the failure of a feedback sensor. Known component failure modes shall be considered in the design of the software and checks designed into the software to detect failures.

E.8.2 Interface Control

SCCSFs and their interfaces to safety-critical hardware shall be controlled at all times. The interface shall be monitored to ensure that erroneous or spurious data does not adversely affect the system, that interface failures are detected, and that the state of the interface is safe during power up, power fluctuations and interruptions, and in the event of system errors or hardware failures.

E.8.3 Decision Statements

Decision statements in safety-critical computing system functions shall not rely on inputs of all ones or all zeros, particularly when this information is obtained from external sensors.

E.8.4 Inter-CPU Communications

Inter-CPU communications shall successfully pass verification checks in both CPUs prior to the transfer of safety-critical data. Periodic checks shall be performed to ensure the integrity of the interface. Detected errors shall be logged. If the interface fails several consecutive transfers, the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

operator shall be alerted and the transfer of safety-critical data will be terminated until diagnostic checks can be performed.

E.8.5 Data Transfer Messages

Data transfer messages shall be of a predetermined format and content. Each transfer shall contain a word or character string indicating the message length (if variable), the type of data, and the content of the message. At a minimum, parity checks and checksums shall be used for verification of correct data transfer. CRCs shall be used where practical. No information from data transfer messages shall be used prior to verification of correct data transfer.

E.8.6 External Functions

External functions requiring two or more safety-critical signals from the software (e.g., arming of an ignition safety device or arm fire device and release of an air launched weapon) shall not receive all of the necessary signals from a single input/output register or buffer.

E.8.7 Input Reasonableness Checks

Limit and reasonableness checks, including time limits and dependencies, shall be performed on all analog and digital inputs and outputs prior to safety-critical functions being executed based on those values. No safety-critical functions shall be executable based on safety-critical analog or digital inputs that cannot be verified.

E.8.8 Full-Scale Representations

The software shall be designed such that the full scale and zero representations of the software are fully compatible with the scales of any digital-to-analog, analog-to-digital, digital-to-synchro, and synchro-to-digital converters.

E.9 Human Interface

The design requirements of this section apply to the design of the human interface to safety-critical computing systems.

E.9.1 Operator/Computing System Interface

The topics herein are typically labeled or addressed by the Human Systems Integration discipline, sometimes referred to as Human Factors Engineering. The following are a list of considerations for safety at the interface of an operator to any system:

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Computer/human interface (CHI) issues
- Displays
- To reduce human errors and upgrade completeness of display information and clarity, SCCSF display data must be duplicated, where possible or practicable, by output designed to be independently generated by a non-software device (e.g., electro-mechanical altimeter with clock-face and needle).
- Hazardous condition alarms and warnings
- Easily distinguished between types of alerts and warnings; corrective action is required to clear
- Process cancellation
- Multiple operator actions to initiate a hazardous function
- Detection of improper operator entries.

E.9.1.1 CHI Issues

CHI issues are a distinct specification and design issue for the system. Many of the CHI functions will be implemented in software, and CHI issues are frequently treated at the same time as software in milestone reviews. Pertinent considerations include:

- Has the developer explicitly addressed the safety-critical aspects of the design of the CHI? Has this included analysis of anticipated single and multiple operator failures? What human factors, ergonomic, and cognitive science analyses were done (e.g., cognitive overload and ambiguity of display information)?
- Does the design ensure that invalid operator requests are flagged and identified as such to the operator (vs. ignoring them or silently mapping them to correct values)?
- Does the developer ensure that the system always requires a minimum of two independent commands to perform a safety-critical function? Before initiating any critical sequence, does the design require operator response or authorization?
- Does the developer ensure that there are no silent mode changes that can put the system in a different safety-significant state without operator awareness (e.g., does the design allow critical mode transitions to happen without notification)?
- Does the developer ensure that there is a positive reporting of changes of safety-critical states?
- Does the system design provide for notification that a safety function has been executed, and is the operator notified of the cause?
- Are all critical inputs clearly distinguished? Are all such inputs checked for range and consistency validity?

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.9.2 Processing Cancellation

The software shall be designed such that the operator may cancel current processing with a single action and have the system revert to a designed safe state. The system shall be designed such that the operator may exit potentially unsafe states with a single action. This action shall revert the system to a known safe state (e.g., the operator shall be able to terminate missile launch processing with a single action that will safe the missile). The action may consist of pressing two keys, buttons, or switches at the same time. Where operator reaction time is not sufficient to prevent a mishap, the software shall revert the system to a known safe state, report the failure, and report the system status to the operator.

E.9.3 Hazardous Function Initiation

Two or more unique operator actions shall be required to initiate any potentially hazardous function or sequence of functions. The actions required shall be designed to minimize the potential for inadvertent actuation and shall be checked for proper sequence.

E.9.4 Safety-Critical Displays

Safety-critical operator displays, legends, and other interface functions shall be clear, concise, unambiguous, and be duplicated (where possible) using separate display devices. Likewise, the clarity shall be compared across all possible displays of all possible data the SoS may display. For example, every typeface, color, and taxonomy shall be consistent across all displays whenever the capability of situational awareness is to be implemented on even one of the displays.

E.9.5 Operator Entry Errors

The software shall be capable of detecting improper operator entries or sequences of entries or operations and prevent execution of safety-critical functions as a result. The software shall alert the operator to the erroneous entry or operation. Alerts shall indicate the error and corrective action. The software shall also provide positive confirmation of valid data entry or actions taken (i.e., the system shall provide visual and/or aural feedback to the operator such that the operator knows that the system has accepted the action and is processing it). The system shall also provide a real-time indication that it is functioning. Processing functions requiring several seconds or longer shall provide a status indicator during processing.

E.9.6 Safety-Critical Alerts

Alerts shall be designed such that routine alerts are readily distinguished from safety-critical alerts. The operator shall not be able to clear a safety-critical alert without taking corrective action or performing subsequent actions required to complete the ongoing operation.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.9.7 Unsafe Situation Alerts

Signals alerting the operator to unsafe situations shall be directed as straightforward as practical to the operator interface.

E.9.8 Unsafe State Alerts

If an operator interface is provided and a potentially unsafe state has been detected, the system shall alert the operator to the anomaly detected, the action taken, and the resulting system configuration and status.

E.10 Critical Timing and Interrupt Functions

The following design requirements and guidelines apply to safety-critical timing functions and interrupts.

E.10.1 Safety-Critical Timing

Safety-critical timing functions shall be controlled by the computer and shall not rely on human input. Safety-critical timing values shall not be modifiable by the operator from system consoles, unless specifically required by the system design. In these instances, the computer shall determine the reasonable timing values.

E.10.2 Valid Interrupts

The software shall be capable of discriminating between valid and invalid external and internal interrupts. Invalid interrupts shall not be capable of creating hazardous conditions. Valid external and internal interrupts shall be defined in system specifications. Internal software interrupts are not a preferred design because they reduce the analyzability of the system.

E.10.3 Recursive Loops

Recursive and iterative loops shall have a maximum documented execution time. Reasonableness checks will be performed to prevent loops from exceeding the maximum execution time.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.10.4 Time Dependency

The results of a program should not be dependent on the time taken to execute the program or the time at which execution is initiated. Safety-critical routines in real-time programs shall ensure that the data used is still valid (e.g., by using senescence checks).

E.11 Software Design and Development Requirements and Guidelines

The requirements and guidelines of this section apply to the design and coding of the software.

E.11.1 Coding Requirements and Issues

The formal and accurate documentation of safety issues involving specific software languages has not been accomplished. However, there are lessons learned and language-specific issues that have been provided that should be reviewed for applicability to a given project where a software language has been selected. Note that these issues are not an endorsement or rejection of any specific language, but a list of considerations that may be beneficial in the identification of safety requirements. Also note that as languages are updated and new versions are provided to the software development community, issues that are identified here may no longer exist in the updated version.

The following applies to the software-coding phase.

- Language issues (e.g., Ada and C++)
- Logic errors
- Cumulative data errors
- Drift in clocks and round-off errors
- Specific features and requirements
- No unused executable code; no unreferenced variables, variable names, and declaration for SCFs; loop entry and exits; use of annotation within code; assignment statements; conditional statements; strong data typing; ban of global variables for SCFs; and safety-critical files
- All safety-critical software to occupy same amount of memory
- Single execution path for safety-critical functions
- No unnecessary or undocumented features
- No bypass of required system functions
- Prevention of runaway feedback loops.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.11.1.1 Ada Language Issues

The Ada programming language provides considerable support for preventing many causes of unpredictable behavior allowed in other languages. For example, unless pragma Suppress or unchecked conversion (and certain situations with pragma Interface) are used, implicit constraint checks prevent the classic C programming bug of writing a value into the 11th element of a 10-element array (overwriting and corrupting an undetermined region of memory with unknown results can be catastrophic). However, the Ada language definition identifies specific rules to be obeyed by Ada programs, but no compile-time or run-time check is required to enforce this. If a program violates one of these rules, the program is said to be erroneous.

According to the language definition, the results of executing an erroneous program are undefined and unpredictable. For example, there is no requirement for a compiler to detect the reading of uninitialized variables or for this error to be detected at run time. If a program does execute such use of uninitialized variables, the effects are undefined. The program could raise an exception (e.g., Program_Error or Constraint_Error), halt, produce a random value in the variable, or the compiler may have a pre-defined value for references to uninitialized variables (e.g., 0). The overall confidence that the program office has in the predictable behavior of the software will be undermined if there are instances of erroneous Ada programs with no evidence provided to show that they do not present a hazard.

There are several other aspects of the use of Ada that can introduce unpredictable behavior, timing, or resource usage, while not strictly erroneous:

- Are all constraints static? If not, how are the following sources of unpredictable behavior shown to prevent a hazard (Constraint_Error)?
- Use of unpredictable memory due to elaboration of non-static declarative items
- For Ada floating point values, are the relational operators "<", '>", ""=, and "/=" precluded? Because of the way floating point comparisons are defined in Ada, the values of the listed operators depend on the implementation. However, "<=" and ">=" do not depend on the implementation. Note that for Ada, floating point it is not guaranteed (e.g., "X <= Y" is the same as "not (X>Y)"). How are floating point operations ensured to be predictable, or how is the lack of predictability shown to not represent a hazard by the developer?
- Does the developer use address clauses? If so, what restrictions are enforced on the address clauses to prevent data overlay attempts which result in an erroneous program?
- If Ada access types are used, has the developer identified all potential problems that can result with access types (e.g., unpredictable memory use, erroneous programs if Unchecked_Deallocation is used and there are references to a deallocated object, aliasing, unpredictable timing for allocation, and constraint checks) and provided evidence that these do not represent hazards?

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- If pragma Interface is used, does the developer ensure that no assumptions about data values are violated in the foreign language code that might not be detected upon returning to the Ada code (e.g., passing a variable address to a C routine that violates a range constraint may not be detected upon return to Ada code, enabling the error to propagate before detection)?
- Does the developer ensure that all out and in out mode parameters are set before returning from a procedure or entry call unless an exception is propagated, or provide evidence that there is no case where returning with an unset parameter (and therefore creating an erroneous program) could introduce a hazard?
- Since Ada supports recursion, has the developer identified restrictions on the use of recursion or otherwise presented evidence that recursion will not introduce a hazard (e.g., through exhaustion of the stack or unpredictable storage timing behavior)?
- Are any steps taken to prevent the accidental reading of an uninitialized variable in the program through coding standards (defect prevention) and code review or static analysis (defect removal)? Does the developer know what the selected compiler's behavior is when uninitialized variables are referenced? Has the developer provided evidence that there are no instances of reading uninitialized variables that introduce a hazard, as such a reference results in an erroneous program?
- If the pre-defined Ada generic function Unchecked_Conversion is used, does the developer ensure that such conversions do not violate the constraints of objects of the result type, as such a conversion results in an erroneous program?
- In Ada, certain record types and private types have discriminants whose values distinguish alternative forms of values of one of these types. Certain assignments and parameter bindings for discriminants result in an erroneous program. If the developer uses discriminants, how does he/she ensure that such erroneous uses do not present a hazard?

E.11.2 Modular Code

Software design and code shall be modular. Modules shall have one entry and one exit point.

E.11.3 Number of Modules

The number of program modules containing safety-critical functions shall be minimized where possible within the constraints of operational effectiveness, computer resources, and good software design practices.

E.11.4 Execution Path

SCCSFs shall have one and only one possible path leading to their execution.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.11.5 Halt Instructions

Halt, stop, or wait instructions shall not be used in code for safety-critical functions. Wait instructions may be used where necessary to synchronize input/output and when appropriate handshake signals are not available.

E.11.6 Single Purpose Files

Files used to store safety-critical data shall be unique and shall have a single purpose. Scratch files used for temporary storage of data during or between processes shall not be used for storing or transferring safety-critical information, data, or control functions.

E.11.7 Unnecessary Features

The operational and support software shall contain only those features and capabilities required by the system. The programs shall not contain undocumented or unnecessary features.

E.11.8 Indirect Addressing Methods

Indirect addressing methods shall be used only in well-controlled applications. When used, the address shall be verified as being within acceptable limits prior to execution of safety-critical operations. Data written to arrays in safety-critical applications shall have the address boundary checked by the compiled code.

E.11.9 Uninterruptible Code

If interrupts are used, sections of the code which have been defined as uninterruptible shall have defined execution times monitored by an external timer.

E.11.10 Safety-Critical Files

Files used to store or transfer safety-critical information shall be initialized to a known state before and after use. Data transfers and data stores shall be audited, where practical, to allow traceability of system functioning.

E.11.11 Unused Memory

All processor memory not used for or by the operational program shall be initialized to a pattern that will cause the system to revert to a safe state if executed. Memory shall not be filled with random numbers, halt, stop, wait, or no-operation instructions. Data or code from previous

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

overlays or loads shall not be allowed to remain. For example, if the processor architecture halts upon receipt of non-executable code, a watchdog timer shall be provided with an interrupt routine to revert the system to a safe state. If the processor flags non-executable code as an error, an error handling routine shall be developed to revert the system to a safe state and terminate processing. Information shall be provided to the operators to alert them to the failure or fault observed and to inform them of the resultant safe state to which the system was reverted.

E.11.12 Overlays of Safety-Critical Software Shall Occupy the Same Amount of Memory

Where less memory is required for a particular function, the remainder shall be filled with a pattern that will cause the system to revert to a safe state if executed. The remainder shall not be filled with random numbers, halt, stop, no-op, wait instructions, or data or code from previous overlays.

E.11.13 Operating System Functions

If an operating system function is provided to accomplish a specific task, operational programs shall use that function and not bypass it or implement it in another fashion.

E.11.14 Compilers

The implementation of software compilers shall be validated to ensure that the compiled code is fully compatible with the target computing system and application (may be done once for a target computing system).

E.11.15 Flags and Variables

Flags and variable names shall be unique. Flags and variables shall have a single purpose and shall be defined and initialized prior to use.

E.11.16 Loop Entry Point

Loops shall have one and only one entry point. Branches into loops shall not be used. Branches out of loops shall lead to a single exit point placed after the loop within the same module.

E.11.17 Software Maintenance Design

The software shall be annotated, designed, and documented for ease of analysis, maintenance, and testing of future changes to the software.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.11.18 Variable Declaration

Variables or constants used by a safety-critical function will be declared and initialized at the lowest possible level.

E.11.19 Unused Executable Code

Operational program loads shall not contain unused executable code.

E.11.20 Unreferenced Variables

Operational program loads shall not contain unreferenced or unused variables or constants.

E.11.21 Assignment Statements

SCCSFs and other safety-critical software items shall not be used in one-to-one assignment statements unless the other variable is also designated as safety-critical (i.e., shall not be redefined as another non-safety-critical variable).

E.11.22 Conditional Statements

Conditional statements shall have all possible conditions satisfied and under full software control (i.e., there shall be no potential unresolved input to the conditional statement). Conditional statements shall be analyzed to ensure that the conditions are reasonable for the task and that all potential conditions are satisfied and not left to a default condition. All condition statements shall be annotated with their purpose and expected outcome for given conditions.

E.11.23 Strong Data Typing

Safety-critical functions shall exhibit strong data typing. Safety-critical functions shall not employ a logic "1" and "0" to denote the safe and armed (potentially hazardous) states. The armed and safe state for munitions shall be represented by at least a unique four-bit pattern. The safe state shall be a pattern that cannot, as a result of a one-, two-, or three-bit error, represent the armed pattern. The armed pattern shall also not be the inverse of the safe pattern. If a pattern other than these two unique codes is detected, the software shall flag the error, revert to a safe state, and notify the operator.

There are additional issues with off-the-shelf software. For example, with respect to safe/arm bit patterns, cases exist where a four-bit pattern is not a compliment, but a three-bit error can still cause the wrong state. That is, if SAFE = 1010 and ARM = 0111, a bit error in the first, second, and fourth positions can cause the wrong state to occur. Safety shall address states, such as arm

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

and safe, to eliminate a three-bit error until assessment shows that the probability of a three-bit error is low enough.

E.11.24 Timer Values Annotated

Values for timers shall be annotated in the code. Comments shall include a description of the timer function, its value, and the rationale or a reference to the documentation explaining the rationale for the timer value. These values shall be verified and shall be examined for reasonableness for the intended function.

E.11.25 Critical Variable Identification

Safety-critical variables shall be identified in such a manner that they can be readily distinguished from non-safety-critical variables (e.g., all safety-critical variables begin with the letter “S”).

E.11.26 Global Variables

Global variables shall not be used for safety-critical functions.

E.12 Software Maintenance Requirements and Guidelines

The requirements and guidelines of this section are applicable to the maintenance of the software in safety-critical computing system applications. The requirement applicable to the design and development phase, as well as the software design and coding phases, are also applicable to the maintenance of the computing system and software.

E.12.1 Critical Function Changes

Changes to SCCSFs on deployed or fielded systems shall be issued as a complete package for the modified unit or module and shall not be patched.

E.12.2 Critical Firmware Changes

When not implemented at the depot level or in manufacturer facilities under appropriate quality control, firmware changes shall be issued as a fully functional and tested circuit card. Design of the card and installation procedures should minimize the potential for damage to the circuits due to mishandling, electrostatic discharge, or normal or abnormal storage environments and shall be accompanied with the proper installation procedures.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.12.3 Software Change Medium

When not implemented at the depot level or in manufacturer facilities under appropriate quality control, software changes shall be issued as a fully functional copy on the appropriate medium. The medium, its packaging, and the procedures for loading the program should minimize the potential damage to the medium due to mishandling, electrostatic discharge, potential magnetic fields, or normal or abnormal storage environments and shall be accompanied with the proper installation procedures.

E.12.4 Modification Configuration Control

All modifications and updates shall be subject to strict configuration control. The use of automated CM tools is encouraged.

E.12.5 Version Identification

Modified software or firmware shall be clearly identified with the version of the modification, including configuration control information. Both physical (e.g., external label) and electronic (e.g., internal digital identification) "fingerprinting" of the version shall be used.

E.13 Software Analysis and Testing

The requirements and guidelines of this section are applicable to the software-testing phase.

E.13.1 General Testing Guidelines

Systematic and thorough testing is required as evidence for critical software assurance; however, testing is necessary but not sufficient. Testing is the primary method for providing evidence about the actual behavior of the software produced; however, the evidence is incomplete because testing for non-trivial systems is a sampling of input states and is not an exhaustive exercise of all possible system states. In addition, many testing and reliability estimation techniques developed for hardware components are not directly applicable to software. Therefore, care must be taken when interpreting the implications of test results for operational reliability.

Testing to provide evidence for critical software assurance differs in emphasis from general software testing to demonstrate correct behavior. Emphasis should be placed on demonstrating that the software does not present a hazard, even under stressful conditions. A considerable amount of testing for critical software should include fault injection, boundary condition and out-of-range testing, and exercising those portions of the input space that are related to potentially hazardous scenarios (e.g., critical operator functions or interactions with safety-critical devices). In commercial aviation, the term "considerable" includes those functions at the Catastrophic level of risk. Confidence in the results of testing is increased when there is evidence that the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

assumptions made in designing and coding the system are not shared by the test developers (i.e., some degree of independence between testers and developers has been maintained). Analysis should consider the following:

- Does the developer provide evidence that critical software testing has addressed not only nominal correctness (e.g., stimulus/response pairs to demonstrate satisfaction of functional requirements), but also robustness in the face of stress? This includes a systematic plan for fault injection, testing boundary and out-of-range conditions, testing the behavior when capacities and rates are extreme (e.g., no input signals from a device for longer than operationally expected or no more frequent input signals from a device than operationally expected), testing error handling (for internal faults), and the identification and demonstration of critical software behavior in the face of the failure of other components.
- Does the developer provide evidence of the independence of test planning, execution, and review for critical software? Are unit tests developed, reviewed, executed, and interpreted by someone other than the individual developer? Has independent test planning and execution been demonstrated at the integration test level?
- Has some amount of independent "free play" testing (where the user tests or operates randomly) been provided? If so, is there evidence during this testing that the critical software is robust in the face of unexpected scenarios and input behavior, or does this independent testing provide evidence that the critical software is fragile (Navy free play testing should place a high priority on exercising the critical aspects of the software and present the system with the kinds of operational errors and stresses that the system will face in the field)?
- Does the developer's software problem tracking system provide evidence that the rate and severity of errors exposed in testing are diminishing as the system approaches operational testing, or is there evidence of "thrashing" and increasing fragility in the critical software? Does the problem tracking system severity classification scheme reflect the potential mishap severity of an error so that evidence of the hazard implications for current issues can be reviewed?
- Has the developer provided evidence that the tests that exercise the system represent a realistic sampling of expected operational inputs? Has testing been dedicated to randomly selected inputs reflecting the expected operational scenarios (this is another way to provide evidence that implicit assumptions in the design do not represent hazards in critical software, since the random inputs will not be selectively screened by implicit assumptions)?

E.13.2 Trajectory Testing for Embedded Systems

There is a fundamental challenge to the amount of confidence that software testing can provide for certain classes of programs. Unlike "memory-less" batch programs that can be completely defined by a set of simple stimulus/response pairs, these programs appear to run continuously. One cannot identify discrete runs, and the behavior at any point may depend on events in the

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

past. In systems where there are major modes or distinct partitioning of the program behavior depending on state, there is mode-remembered data that is retained across mode-changes. The key issue for assurance is the extent to which these characteristics have been reflected in the design and testing of the system. If these characteristics are ignored and the test set is limited to a simple set of stateless stimulus/response pairs, the extrapolation to the operational behavior of the system is weakened. Questions for consideration include:

- Has the developer identified the sensitivities to persistently stale data (especially publish/subscribe architectures and middleware) and the input trajectory the system is expected to experience? Is this reflected in the test plans and test descriptions?
- Are the developer's assumptions about prohibited or impossible trajectories and mode changes explicit with respect to critical functions? For example, there is a danger that the model used to determine impossible trajectories overlooks the same situation overlooked by the programmer who introduced a serious bug. It is important that any model used to eliminate impossible trajectories be developed independently of the program. Most safety experts would feel more comfortable if some tests were conducted with "crazy" trajectories.

E.13.3 Formal Test Coverage

All safety-significant software testing shall be conducted in accordance with a formal test coverage document and include code coverage analysis with adequate documentation of test results. Computer-based tools shall be used to ensure that the coverage is as complete as possible.

E.13.4 Go/No-Go Path Testing

Software testing shall include GO/NO-GO path testing.

E.13.5 Input Failure Modes

Software testing shall include hardware and software input failure mode testing.

E.13.6 Boundary Test Conditions

Software testing shall include boundary, out-of-bounds, and boundary crossing test conditions.

E.13.7 Input Rata Rates

Software testing shall include minimum and maximum input data rates in worst case configurations to determine the system's capabilities and responses to these conditions.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

E.13.8 Zero Value Testing

Software testing shall include input values of zero, zero crossing, approaching zero from either direction, and similar values for trigonometric functions.

E.13.9 Regression Testing

SCCSFs in which changes have been made shall be subject to complete regression testing, along with all associated start-up, run-time, and maintenance phase expected trajectories.

E.13.10 Operator Interface Testing

Operator interface testing shall include operator errors during safety-critical operations to verify safe system response to these errors.

E.13.11 Duration Stress Testing

Software testing shall include duration stress testing. The stress test time shall be continued for at least the maximum expected operating time for the system. Testing shall be conducted under simulated operational environments. Additional stress duration testing should be conducted to identify potential critical functions (e.g., timing, data senescence, and resource exhaustion) that are adversely affected as a result of operational duration. Software testing shall include throughput stress testing (e.g., CPU, data bus, memory, and input/output) under peak loading conditions.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

APPENDIX F LESSONS LEARNED

F.1 Therac Radiation Therapy Machine Fatalities

F.1.1 Summary

Eleven Therac-25 therapy machines were operationally installed, five in the United States and six in Canada. The Canadian Crown (Government owned) company Atomic Energy of Canada Limited manufactured the machines. The -25 model was an advanced model over earlier models (-6 and -20 models, corresponding to energy delivery capacity) with more energy and automation features. Although all models had some software control, the -25 model had many new features and had replaced most of the hardware interlocks with software versions. There was no record of any malfunctions resulting in patient injury from any of the earlier models (earlier than the -25). The software control was implemented in a DEC model PDP 11 processor using a custom executive and assembly language. A single programmer implemented virtually all of the software. The programmer had an unknown level of formal education and produced very little documentation of the software.

Between June 1985 and January 1987, there were six known accidents involving massive radiation overdoses by the Therac-25; three of the six resulted in fatalities. The company did not respond effectively to early reports, citing the belief that the software could not be a source of failure. Records show that software was deliberately left out of an otherwise thorough safety analysis performed in 1983, which used fault tree methods. Software was excluded because "software errors have been eliminated because of extensive simulation and field testing. (Also) software does not degrade due to wear, fatigue, or reproduction process." Other types of software failures were assigned very low failure rates with no apparent justification. After a large number of lawsuits and extensive negative publicity, the company decided to withdraw from the medical instrument business and concentrate on its main business of nuclear reactor control systems.

The accidents were due to several design deficiencies involving a combination of software design defects and system operational interaction errors. There were no apparent review mechanisms for software design or quality control. The continuing recurrence of the accidents before effective corrective action was taken was a result of management's view. This view had faith in the correctness of the software without any apparent evidence to support it. The errors were not discovered because the policy was to fix the symptoms without investigating the underlying causes, of which there were many.

F.1.2 Key Facts

- The software was assumed to be fail-safe and was excluded from normal safety analysis review

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- The software design and implementation had no effective review or quality control practices
- Software testing at all levels was obviously insufficient, given the results
- Hardware interlocks were replaced by software without supporting safety analysis
- There was no effective reporting mechanism for field problems involving software
- Software design practices (contributing to the accidents) did not include basic shared data and contention management mechanisms normally found in multi-tasking software. The necessary conclusion is that the programmer was not fully qualified for the task
- The design was unnecessarily complex for the problem. For example, there were more parallel tasks than necessary. This was a direct cause of some of the accidents.

F.1.3 Lessons Learned

- Changeover from hardware to a software implementation must include a review of assumptions, physics, and rules
- Testing should include possible abuse or bypassing of expected procedures
- Design and implementation of software must be subject to the same safety analysis, review, and quality control as other parts of the system
- Hardware interlocks should not be completely eliminated when incorporating software interlocks
- Programmer qualifications are as important as qualifications for any other member of the engineering team.

F.2 Missile Launch Timing Error Causes Hang-Fire

F.2.1 Summary

An aircraft was modified from a hardware-controlled missile launcher to a software-controlled launcher. The aircraft was properly modified according to standards, and the software was fully tested at all levels before delivery to operational test. The normal weapons rack interface and safety overrides were fully tested and documented. The aircraft was loaded with a live missile (with an inert warhead) and sent out onto the range for a test firing.

The aircraft was commanded to fire the weapon, whereupon it did as designed. Unfortunately, the design did not specify the amount of time to unlock the holdback and was coded to the assumption of the programmer. In this case, the assumed time for unlock was insufficient and the holdback locked before the weapon left the rack. As the weapon was powered, the engine drove the weapon while attached to the aircraft. This resulted in a loss of altitude and a wild ride, but the aircraft landed safely with a burned out weapon.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

F.2.2 Key Facts

- Proper process and procedures were followed as far as specified
- The product specification was re-used without considering differences in the software implementation (e.g., the timing issues). Hence, the initiating event was a specification error.
- While the acquirer and user had experience with the weapons system, neither had experience in software. Also, the programmer did not have experience with the details of the weapons system. The result was that the interaction between the two parts of the system was not understood by any of the parties.

F.2.3 Lessons Learned

- Because the software-controlled implementation was not fully understood, the result was flawed specifications and incomplete tests. Therefore, even though the software and subsystem were thoroughly tested against the specifications, the system design was in error and a mishap occurred.
- Changeover from hardware to software requires a review of design assumptions by all relevant specialists acting jointly. This joint review must include all product specifications, interface documentation, and testing.
- The test, verification, and review processes must each include end-to-end event review and test.

F.3 Reused Software Causes Flight Controls to Shut Down

F.3.1 Summary

A research vehicle was designed with fly-by-wire digital control and, for research and weight considerations, had no hardware backup systems installed. The normal safety and testing practices were minimized or eliminated by citing many arguments. These arguments cited use of experienced test pilots, limited flight and exposure times, minimum number of flights, controlled airspace, use of monitors and telemetry, etc. The argument also justified the action as safer because the system reused software from similar currently operational vehicles.

The aircraft flight controls went through every level of test, including “iron bird” laboratory tests that allow direct measurement of the response of the flight components. The failure occurred on the flight line the day before actual flight was to begin after the system had successfully completed all testing. For the first time, the flight computer was operating unrestricted by test routines and controls. A reused portion of the software was inhibited during earlier testing because it conflicted with certain computer functions. This was part of the reused software taken

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

from a proven and safe platform because of its functional similarity. This portion was now enabled and running in the background.

Unfortunately, the reused software shared computer data locations with certain safety-critical functions and was not partitioned nor checked for valid memory address ranges. The result was that as the flight computer functioned for the first time, it used data locations where this reused software had stored out-of-range data on top of safety-critical parameters. The flight computer then performed according to its design when detecting invalid data and reset itself. This happened sequentially in each of the available flight control channels until there were no functioning flight controls. Since the system had no hardware backup system, the aircraft would have stopped flying if it were airborne. The software was quickly corrected and was fully operational in the following flights.

F.3.2 Key Facts

- Proper process and procedures were minimized for apparently valid reasons (e.g., the (offending) software was proven by its use in other similar systems)
- Reuse of the software components did not include review and testing of the integrated components in the new operating environment. In particular, memory addressing was not validated with the new programs that shared the computer resources.

F.3.3 Lessons Learned

- Safety-critical, real-time flight controls must include full integration testing of end-to-end events. In this case, the reused software should have been functioning within the full software system.
- Arguments to bypass software safety, especially in software containing functions capable of a Kill/Catastrophic event, must be reviewed at each phase. Several of the arguments to minimize software safety provisions were compromised before the detection of the defect.

F.4 Flight Controls Fail at Supersonic Transition

F.4.1 Summary

A front line aircraft was rigorously developed, thoroughly tested by the manufacturer, and exhaustively tested by the Government and by the using DoD Service. Dozens of aircraft had been accepted and were operational worldwide when the DoD Service asked for an upgrade to the weapons systems. One particular weapon test required significant telemetry. The aircraft change was again developed and tested to the same high standards, including nuclear weapons carriage clearance. This additional testing data uncovered a detail missed in all of the previous testing.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The telemetry showed that the aircraft computers all failed—ceased to function and then restarted—at specific airspeed (Mach 1). The aircraft had sufficient momentum and mechanical control of other systems so that it effectively coasted through this anomaly, and the pilot did not notice.

The cause of this failure originated in the complex equations from the aerodynamicist. His specialty assumes the knowledge that this particular equation will asymptotically approach infinity at Mach 1. The software engineer does not inherently understand the physical science involved in the transition to supersonic speed at Mach 1. The system engineer who interfaced between these two engineering specialists was not aware of this assumption, and after receiving the aerodynamicist's equation for flight, forwarded the equation to software engineering for coding. The software engineer did not plot the equation and merely encoded it in the flight control program.

F.4.2 Key Facts

- Proper processes and procedures were followed to the stated requirements
- The software specification did not include the limitations of the equation describing a physical science event
- The computer hardware accuracy was not considered in the limitations of the equation
- The various levels of testing did not validate the computational results for the Mach 1 portion of the flight envelope.

F.4.3 Lessons Learned

- Specified equations describing physical world phenomenon must be thoroughly defined, with assumptions as to accuracy, ranges, use, environment, and limitations of the computation
- When dealing with requirements that interface between disciplines, it must be assumed that each discipline knows little or nothing about the other, and therefore must include basic assumptions
- Boundary assumptions should be used to generate test cases because the more subtle failures caused by assumptions are not usually covered by ordinary test cases (division by zero, boundary crossing, singularities, etc.).

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

F.5 Incorrect Missile Firing from Invalid Setup Sequence

F.5.1 Summary

A battle command center with a network controlling several missile batteries was operating in a field game exercise. As the game advanced, an order to reposition the battery was issued to an active missile battery. This missile battery disconnected from the network, broke down their equipment, and repositioned to a new location in the grid.

The repositioned missile battery arrived at the new location and commenced set up. A final step was connecting the battery into the network. This was allowed in any order. The battery personnel were still occupying the erector/launcher when the connection that attached the battery into the network was made elsewhere on the site. This cable connection immediately allowed communication between the battery and the battle command center.

The battle command center, meanwhile, had prosecuted an incoming hostile and designated the battery to fire, but targeted to use the old location of the battery. As the battery was off-line, the message was buffered. Once the battery crew connected the cabling, the battle command center computer sent the last valid commands from the buffer and the command was immediately executed. Personnel on the erector/launcher were thrown clear as the erector/launcher activated on the old slew and acquire command. Personnel injury was slight as no one was pinned or impaled when the erector/launcher slewed.

F.5.2 Key Facts

- Proper process and procedures were followed as specified
- Subsystems were developed separately with interface control documents
- Messages containing safety-critical commands were not aged and reassessed once buffered
- Battery activation was not inhibited until personnel had completed the set-up procedure.

F.5.3 Lessons Learned

- System engineering must define the sequencing of the various states (dismantling, reactivating, shutdown, etc.) of all subsystems with human confirmations and re-initialization of state variables (e.g., site location) at critical points
- System integration testing should include buffering messages (particularly safety critical) and demonstration of disconnect and restart of individual subsystems to verify that the system always safely transitions between states

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Operating procedures must clearly describe (and require) a safe and comprehensive sequence for dismantling and reactivating the battery subsystems, with particular attention to the interaction with the network.

F.6 Operator's Choice of Weapon Release Overridden by Software Control

F.6.1 Summary

During field practice exercises, a missile weapon system was carrying both practice and live missiles to a remote site and was using the transit time for slewing practice. Practice and live missiles were located on opposite sides of the vehicle. The acquisition and tracking radar was located between the two sides, causing a known obstruction to the missiles' field of view.

While correctly following command-approved procedures, the operator acquired the willing target, tracked it through various maneuvers, and pressed the weapons release button to simulate firing the practice missile. Without the knowledge of the operator, the software was programmed to override his missile selection in order to present the best target to the best weapon. The software noted that the current maneuver placed the radar obstruction in front of the practice missile seeker, while the live missile had acquired a positive lock on the target and was unobstructed. The software, therefore, optimized the problem and deselected the practice missile and selected the live missile. When the release command was sent, it went to the live missile and "missile away" was observed from the active missile side of the vehicle when no launch was expected.

The friendly target had been observing the maneuvers of the incident vehicle and noted the unexpected live launch. Fortunately, the target pilot was experienced and began evasive maneuvers, but the missile tracked and still detonated in close proximity.

F.6.2 Key Facts

- Proper procedures were followed as specified, and all operations were authorized
- All operators were thoroughly trained in the latest versions of software
- The software had been given authority to select "best" weapon, but this characteristic was not communicated to the operator as part of training
- The indication that another weapon had been substituted (live vs. practice) by the software was displayed in a manner not easily noticed among other dynamic displays.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

F.6.3 Lessons Learned

- The versatility (and resulting complexity) demanded by the requirement was provided as specified. This complexity, combined with the possibility that the vehicle would employ a mix of practice and live missiles, was not considered. This mix of missiles is a common practice, and system testing must include known scenarios such as this example to find operational hazards.
- Training must describe safety-significant software functions, such as the possibility of software overrides to operator commands. This must also be included in operating procedures available to all users of the system.

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

APPENDIX G EXAMPLE REQUEST FOR PROPOSAL AND STATEMENT OF WORK

G.1 Sample RFP

The following represents a sample system safety paragraph within a Request for Proposal that should be considered as a starting point and is open for negotiation for a given program. As with any example, this sample RFP paragraph must be carefully read and considered as to whether it meets the safety goals and objectives of the program under consideration. In most cases, this language must be tailored.

Suggested Language for Section L, Instructions to Offerors:

System and software safety requirements –

> Offerors shall describe the proposed system and software safety engineering process, comparing it to the elements in MIL-STD-882 or any other regulatory, acceptance, or certification authority. The process will explain the associated tasks that will be accomplished to identify, track, assess, and eliminate hazards as an integral part of the design engineering function. It will also describe the proposed process to reduce residual safety risk to a level acceptable to program management. It will specifically addresses (as a minimum) the role of the proposed system and software safety approach in design, development, management, manufacturing planning, and key program events (as applicable) throughout the system lifecycle.

Suggested Language for Section M, Evaluation Factors for Award:

The offeror's approach will be evaluated based on:

- The acceptability of the proposed system and software safety approach in comparison to the System Safety Program guidance described in MIL-STD-882 and any other regulatory, acceptance, or certification compliance defined as applied to satisfy program objectives
- The effectiveness of the proposed approach that either mitigates or reduces hazard risks to the extent to which:
    - The proposed approach reflects the integration of system and software safety engineering methodologies, processes, and tasks into the planning for this program

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- The proposed approach evaluates the safety impacts of using COTS, GOTS, and NDI hardware and software on the proposed system for this program
- The proposed approach demonstrates the ability to identify, document, track, analyze, and assess system and subsystem-level hazards and their associated causal factors through detailed analysis techniques. The detailed analysis must consider hardware, software, and human interfaces as potential hazard causes.
- The proposed approach communicates initial and derived safety requirements to the design team, including the activities necessary to functionally derive safety requirements from the detailed causal factor analysis
- The proposed approach produces the engineering evidence of hazard elimination or risk abatement to acceptable levels of residual safety risk that balances mishap severity with probability of occurrence
- The proposed approach considers all requirements to meet or obtain the necessary certifications or certificate criteria to field, test, and operate the system.

G.2 Sample Statement of Work

The following example represents sample system safety and software safety paragraphs that can be included in an SOW to ensure that the developer considers and proposes a system safety program to meet program objectives. As with the sample RFP paragraph above, the SOW system safety paragraphs must be considered a starting point for consideration and a negotiation for tailoring. All safety-significant SOW paragraphs must be assessed and tailored to ensure they specify all necessary requirements to meet the safety goals and objectives of the acquiring agency.

G.2.1 System Safety

The following paragraph represents an SOW example where a full-blown system safety program is required which incorporates all functional components of the system, including system-level and subsystem-level hardware, software, and the human element. The suggested language for system safety engineering is as follows:

System Safety

The contractor shall conduct a system safety management and engineering program using MIL-STD-882 as guidance. The program shall include the necessary planning, coordinating, and engineering analysis to:

- Identify the safety-significant functions (safety critical and safety related) of the system and establish a protocol of analysis, design, test, and verification and validation of those functions

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

- Tailor and communicate generic or initial software safety requirements or constraints to the system and software designers as early in the lifecycle as possible
- Identify, document, and track system and subsystem-level hazards
- Identify the system-level effects of each identified hazard
- Categorize each identified hazard in terms of severity and probability of occurrence (specify qualification or quantification of likelihood)
- Conduct in-depth analysis to identify each failure pathway and associated causal factors. This analysis will be to the functional depth necessary to identify logical, practical, and cost-effective mitigation techniques and requirements for each failure pathway initiator (causal factor). This analysis shall consider all hardware, software, and human factor interfaces as potential contributors.
- Derive safety-specific hazard mitigation requirements to eliminate or reduce the likelihood of each causal factor
- Provide engineering evidence (through appropriate inspection, analysis, and test) that each mitigation safety requirement is implemented within the design, and the system functions as required to meet safety goals and objectives
- Conduct a safety assessment of residual safety risk after all design, implementation, and test activities are complete
- Conduct a safety impact analysis on all Software Change Notices or ECPs for engineering baselines under configuration management
- Submit for approval to the certifying authority all waivers and deviations where the system does not meet the safety requirements or certification criteria
- Submit for approval to the acquiring authority an integrated system safety schedule that supports the program's engineering and programmatic milestones.

The results of all safety engineering analysis performed shall be formally documented in a closed-loop hazard tracking database system. The information shall be correlated in such a manner that it can be easily and systematically extracted from the database to produce the necessary deliverable documentation (e.g., FHA, PHA, SRA, SSHA, SHA, O&SHA, FMEA, etc.), as required by the contract. The maturity of the safety analysis shall be commensurate with the maturity of system design in accordance with the acquisition lifecycle phase.

G.2.2 Software Safety

The following example represents a sample software safety program as a stand-alone task(s) where another contractor or agency possesses the responsibility of system safety engineering. The software safety program is required to incorporate all functional and supporting software of the system which has the potential to influence system-level and subsystem-level hazards. The suggested language for software safety engineering program is as follows:

Software Safety

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

The contractor shall conduct a software safety engineering program using MIL-STD-882 as guidance. This program shall fully support the existing system safety engineering program and functionally link software architecture to hazards and their failure pathways. The program shall include the necessary planning, coordinating, and engineering analysis to:

- Identify the safety significant functions (safety critical and safety related) of the system and establish a protocol of analysis, design, test, and verification and validation for those functions within the software development activities
- Establish a software criticality assessment with a level of rigor protocol for the requirements, design, code, and test of safety-significant software functions
- Tailor and communicate generic or initial software safety requirements or constraints to the system and software designers as early in the lifecycle as possible
- Analyze the existing documented hazards to determine software influence on these hazards in terms of causal initiation or causal propagation
- Consider the system-level effects of each identified hazard
- Provide input to system safety engineering as to the potential contributions or implications of the software that would affect probability of occurrence
- Conduct in-depth analysis to identify each failure pathway and associated software causal factors. This analysis will be to the functional depth necessary to identify logical, practical, and cost-effective mitigation techniques and requirements for each failure pathway initiator (causal factor). This analysis shall consider all potential hardware, software, and human factor interfaces as potential contributors.
- Derive safety-specific hazard mitigation requirements to eliminate or reduce the likelihood of each causal factor within the software functional architecture
- Provide engineering evidence (through appropriate inspection, analysis, and test) that each mitigation software safety requirement is implemented within the design, and the system functions as required to meet the stated level of rigor, safety goals, and objectives of the program
- Conduct a safety assessment of all residual safety risk after all design, implementation, and test activities are complete
- Conduct a safety impact analysis on all Software Change Notices, PTRs, or ECPs for engineering baselines under configuration management
- Submit for approval to the acceptance or certifying authority all waivers and deviations where the system does not meet the safety requirements or certification criteria
- Submit for approval to the acquiring authority an integrated system safety schedule that supports the program engineering and programmatic milestones.

The results of all software safety engineering analysis performed shall be formally documented in a closed-loop hazard tracking database system. The information shall be correlated in such a manner that it can be easily and systematically extracted from the database to produce the necessary deliverable documentation (e.g., PHA, SRA, SSHA, SHA, O&SHA, FMEA, etc.), as

Downloaded from http://www.everyspec.com on 2012-03-01T17:53:05.

required by the contract. The maturity of the software safety analysis shall be commensurate with the maturity of system design in accordance with the acquisition lifecycle phase. All software safety analysis shall be conducted and made available to support the goals, objectives, and schedule of the parent system safety program.