

AEROSPACE REPORT NO.  
ATR-2011(8404)-11

# The Elements of an Effective Software Development Plan – Software Development Process Guidebook

November 11, 2011

Marvin C. Gechman  
Software Engineering Subdivision  
Computers and Software Division

Contributing Author:  
Suellen Eslinger  
Software Engineering Subdivision  
Computers and Software Division

Prepared for:  
Space and Missile Systems Center  
Air Force Space Command  
483 N. Aviation Blvd.  
El Segundo, CA 90245-2808

Authorized by: Engineering and Technology Group

**Distribution Statement:** Public release is authorized; distribution unlimited.

201204 20230

AEROSPACE REPORT NO.  
ATR-2011(8404)-11

# The Elements of an Effective Software Development Plan – Software Development Process Guidebook

November 11, 2011

Marvin C. Gechman  
Software Engineering Subdivision  
Computers and Software Division

Contributing Author:  
Suellen Eslinger  
Software Engineering Subdivision  
Computers and Software Division

Prepared for:  
Space and Missile Systems Center  
Air Force Space Command  
483 N. Aviation Blvd.  
El Segundo, CA 90245-2808

Authorized by: Engineering and Technology Group

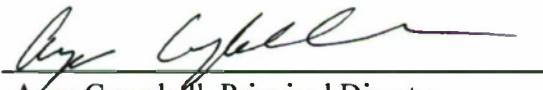
**Distribution Statement:** Public release is authorized; distribution unlimited.



AEROSPACE REPORT NO.  
ATR-2011(8404)-11

# The Elements of an Effective Software Development Plan – Software Development Process Guidebook

Approved by:



Asya Campbell, Principal Director  
Software Engineering Subdivision  
Computers and Software Division  
Engineering and Technology Group

All trademarks, service marks, and trade names are the property of their respective owners.

SP0036(1, 5840, 219, MLM)

## **Abstract**

Every software development program must have a Software Development Plan (SDP). The SDP is required by the software development standards, it is prepared by the contractor, and it is usually submitted with their proposal. The SDP is the vehicle by which the contractor, responsible for software development, documents how the software will be designed, developed, integrated, tested and managed.

The principal objectives of this SDP Guidebook are to: (1) Assist the acquisition agency in evaluating SDPs during source selection and during subsequent update deliveries; (2) Provide guidance to contractors in developing and updating their SDP; and (3) provide a convenient source of reference, during conduct of a software development program, documenting the systematic steps of the process during the full software development life cycle. The Guidebook contains examples and recommended contents of a program-level SDP for large software development efforts.



## Foreword

A poorly planned software development effort is likely to fail—that makes the SDP a critically important software management tool for both large and small software development efforts.

An incomplete or inadequate SDP is a clear red flag. Contractors with a deficient SDP, who are awarded a contract, have historically a high probability of cost and schedule overruns. This Guidebook is intended to significantly increase the probability of a successful software-intensive contract. The principal objectives of this SDP Guidebook are:

- To assist the acquisition agency in evaluating SDPs during source selection and during subsequent updated deliveries of the SDP
- To provide guidance to contractors in preparing and updating their SDPs
- To provide a convenient source of reference, during conduct of a software development program, describing the systematic steps of the software development process during the full software development lifecycle.

The contractor-developed SDP must address at least the following software development concerns:

- What specific process will be followed for software requirements analysis, design, coding, testing, integration, and qualification?
- Who is responsible for each software development task and what is their reporting chain?
- How will software development be managed and with what controls?
- What is the software development schedule and what are the reportable milestones?
- How will management know if the current software project is consistent with planned schedules?
- What documentation will be produced, in what format, and when?
- What standards, practices, and guidelines will be followed and how will they be enforced?
- What reviews will take place, who are the attendees, and when will they take place?
- How will compliance with the SDP be assured?
- What methods will be employed to identify and mitigate software risks?
- How are software development responsibilities managed and flowed down to subcontractors?
- What development and testing support software, environment, and tools are required?
- What is the process for ensuring systematic testing of the developed software?
- What software management measurements (metrics) are planned and what is the process for collection, reporting, analysis and corrective action?
- What process and methods will be used to ensure the quality of the software product?
- How will errors be detected, documented and corrected?
- What software products will be subject to formal configuration management and when?
- What software is deliverable to the acquisition agency and what are the transition plans?
- How will classified data and products be controlled?

Although software planning is performed throughout the software lifecycle, strategic planning up-front usually makes the difference between success and failure of a software development program. The quality and attention to detail in the SDP are major source selection evaluation criteria. A good SDP, at the start of a program, builds the foundation for the teamwork and disciplined trust vital to software lifecycle cooperation and success. The existence of a comprehensive SDP does not guarantee project success. However, a poor SDP at the start of a program is essentially a guarantee of serious problems ahead.

## SDP Guidebook Reading Recommendations

Because of the comprehensive nature of this Guidebook, it is expected that individual sections will be used as a reference, when needed, as opposed to assuming the reader will always read the entire Guidebook. However, it is highly recommended that all users of this Guidebook read, at a minimum, the six-page Introduction of Part 1 and paragraph 1.2.3 of Part 2 describing the software classes and categories referred to throughout Sections 4 and 5 of Part 2.

If the user is planning to read any of the sections dealing with software integration and testing (subsections 5.7 through 5.11), it is highly recommended that they begin by reviewing subsection 3.7 as it provides an overview of the software integration, testing, and verification process described in more detail in subsections 5.7 through 5.11.

To facilitate the lookup of specific topics of interest in this Guidebook, a Subject Index is included in Part 3 as Additional Guidebook Information AGI-5 of this Guidebook. It refers to the subsection, paragraph or subparagraph where the subject is addressed.





## Contents

Abstract .....	iii
Foreword .....	v
SDP Guidebook Reading Recommendations .....	vii
<b>Part 1. SDP Guidebook Introduction .....</b>	<b>1-1</b>
1. Scope and Perspective.....	1-1
2. Organization of this SDP Guidebook .....	1-1
3. Electronic Data Management .....	1-2
4. Example Text and Highlights.....	1-2
5. Terms and Acronyms Used .....	1-2
6. Format of the Process Descriptions .....	1-3
7. Integrated Product Teams (IPTs).....	1-4
8. Analysis and Design Methodologies .....	1-4
9. Format Options for the SDP.....	1-4
10. Tailoring of the SDP .....	1-5
11. Large Versus Small Software Developments .....	1-5
12. Recommended SDP Numbering Format.....	1-6
<b>Part 2. Recommended Contents of a Program-Level SDP for Large Software Development Efforts .....</b>	<b>1-1</b>
1. Scope.....	2-1
1.1 Identification.....	2-1
1.2 System Overview .....	2-2
1.2.1 System Architecture Overview .....	2-2
1.2.2 Software Architecture Overview.....	2-3
1.2.3 Software Classes and Categories.....	2-4
1.3 Document Overview.....	2-7
1.3.1 SDP Component Parts .....	2-7
1.3.2 SDP Organization.....	2-8
1.3.3 SDP Updates .....	2-8
1.4 Relationship to Other Plans .....	2-9
2.1 Government Documents .....	2-11
2.1.1 Government Referenced Documents—Example .....	2-11
2.1.2 Government Applicable Documents—Example .....	2-11
2.2 Non-Government Documents .....	2-11
2.2.1 Non-Government Referenced Documents—Example.....	2-11
2.2.2 Non-Government Applicable Documents—Example .....	2-11
3. Overview of Required Work .....	2-13
3.1 System Acquisition Lifecycle .....	2-13
3.2 Software Requirements and Constraints.....	2-13
3.3 Software Item Overview.....	2-14

3.4	Required Software Lifecycle Activities .....	2-15
3.5	Software Process Overview .....	2-15
3.6	Software Documentation Requirements and Constraints.....	2-16
3.7	Requirements and Constraints on Development Strategy .....	2-19
3.7.1	Development Strategy Factors .....	2-19
3.7.2	Software Integration, Testing, and Verification Approach .....	2-19
3.7.3	Software Integration, Testing, and Verification Objectives.....	2-20
3.7.4	Software Integration, Testing, and Verification Process .....	2-20
3.8	Requirements and Constraints on Schedule and Resources .....	2-20
3.9	Other Requirements and Constraints .....	2-20
4.	General Requirements .....	2-23
4.1	Software Development Process .....	2-23
4.1.1	Mission Critical Software Development Process .....	2-23
4.1.2	Support Software Development Process.....	2-26
4.1.3	Iterative Process .....	2-26
4.2	General Requirements for Software Development .....	2-27
4.2.1	Software Development Methods .....	2-27
4.2.2	Standards for Software Products .....	2-28
4.2.3	Traceability .....	2-29
4.2.4	Reusable Software Products.....	2-30
4.2.5	Assurance of Critical Requirements .....	2-33
4.2.6	Computer Hardware Resource Utilization .....	2-37
4.2.7	Recording Rationale for Key Technical Decisions.....	2-37
4.2.8	Access for Acquirer Review .....	2-37
4.2.9	Software Data Management (Recommended Optional Addition).....	2-38
4.2.10	Software Plans and Work Products (Recommended Optional Addition) .....	2-39
5.	Detailed Requirements .....	2-43
5.1	Project Planning and Oversight .....	2-43
5.1.1	Software Development Planning .....	2-44
5.1.2	Software Item Test Planning .....	2-49
5.1.3	System Test Planning .....	2-50
5.1.4	Planning for Software Transition to Operations .....	2-50
5.1.5	Planning for Software Transition to Maintenance .....	2-51
5.1.6	Following and Updating Plans .....	2-51
5.2	Establishing a Software Development Environment .....	2-51
5.2.1	Software Engineering Environment .....	2-51
5.2.2	Software Integration and Test Environment .....	2-52
5.2.3	Software Development Libraries.....	2-53
5.2.4	Software Development Files .....	2-55
5.2.5	Non-Deliverable Software .....	2-56
5.3	System/Segment Requirements Analysis.....	2-56
5.3.1	Analysis of User Input .....	2-57
5.3.2	Operational Concept .....	2-57
5.3.3	System/Segment Requirements .....	2-58
5.4	System/Segment Design .....	2-58
5.4.1	System-wide/Segment-wide Design Decisions .....	2-60
5.4.2	System/Segment Architectural Design .....	2-61
5.5	Software Requirements Analysis.....	2-62
5.6	Software Design .....	2-68

5.6.1	Software Item-wide Design Decisions .....	2-70
5.6.2	Software Item Architectural Design .....	2-71
5.6.3	Software Item Detailed Design .....	2-73
5.7	Software Implementation and Unit Testing .....	2-77
5.7.1	Software Implementation.....	2-79
5.7.2	Preparing for Unit Testing .....	2-81
5.7.3	Performing Unit Testing .....	2-81
5.7.4	Unit Testing Revision and Retesting .....	2-82
5.7.5	Analyzing and Recording Unit Test Results .....	2-82
5.8	Unit Integration and Testing .....	2-82
5.8.1	Preparing for UI&T .....	2-86
5.8.2	Performing UI&T .....	2-87
5.8.3	UI&T Revision and Retesting.....	2-87
5.8.4	Analyzing and Recording UI&T Results .....	2-88
5.9	Software Item Qualification Testing .....	2-88
5.9.1	Independence in Software Item Qualification Testing .....	2-92
5.9.2	Testing on the Target Computer System .....	2-92
5.9.3	Preparing for Software Item Qualification Testing .....	2-93
5.9.4	Dry Run of Software Item Qualification Testing .....	2-94
5.9.5	Performing Software Item Qualification Testing .....	2-95
5.9.6	SIQT Revision and Retesting.....	2-96
5.9.7	Analyzing and Recording SIQT Results.....	2-97
5.10	Software/Hardware Item Integration and Testing .....	2-98
5.10.1	Preparing For SI/HI Integration and Testing .....	2-100
5.10.2	Performing SI/HI Integration and Testing .....	2-102
5.10.3	Analyzing and Recording SI/HI Integration and Test Results .....	2-103
5.10.4	SI/HI I&T Revision and Retesting .....	2-103
5.11	System Qualification Testing.....	2-104
5.11.1	Independence in System Qualification Testing.....	2-106
5.11.2	Testing On the Target Computer System .....	2-106
5.11.3	Preparing For System Qualification Testing.....	2-106
5.11.4	Dry Run of System Qualification Testing .....	2-107
5.11.5	Performing System Qualification Testing .....	2-107
5.11.6	Analyzing and Recording System Qualification Test Results .....	2-107
5.11.7	System Qualification Testing Revision and Retesting .....	2-108
5.12	Preparing for Software Transition to Operations .....	2-108
5.12.1	Preparing the Executable Software .....	2-109
5.12.2	Preparing Version Descriptions for User Sites .....	2-109
5.12.3	Preparing User Manuals .....	2-109
5.12.4	Installation at User Sites .....	2-111
5.13	Preparing For Software Transition to Maintenance .....	2-111
5.13.1	Preparing the Executable Software .....	2-112
5.13.2	Preparing Source Files .....	2-113
5.13.3	Preparing Version Descriptions for the Maintenance Site.....	2-113
5.13.4	Preparing the "As Built" Software Item Design and Related Information .....	2-113
5.13.5	Updating the System/Subsystem Design Description .....	2-113
5.13.6	Updating the Software Requirements.....	2-114
5.13.7	Updating the System Requirements .....	2-114
5.13.8	Preparing Maintenance Manuals .....	2-114
5.13.9	Transition to the Designated Maintenance Site.....	2-115



5.14	Software Configuration Management.....	2-115
5.14.1	Configuration Identification.....	2-117
5.14.2	Configuration Control.....	2-118
5.14.3	Configuration Status Accounting.....	2-120
5.14.4	Configuration Audits.....	2-120
5.14.5	Packaging, Storage, Handling, and Delivery.....	2-121
5.15	Software Peer Reviews and Product Evaluations.....	2-122
5.15.1	Software Peer Reviews.....	2-123
5.15.2	Software Product Evaluations.....	2-126
5.16	Software Quality Assurance.....	2-127
5.16.1	Software Quality Assurance Evaluations.....	2-128
5.16.2	Software Quality Assurance Records, Including Items to Be Recorded.....	2-129
5.16.3	Independence in Software Quality Assurance.....	2-129
5.16.4	Software Quality Assurance Non-Compliance Issues.....	2-130
5.17	Corrective Action.....	2-130
5.17.1	Problem/Change Reports.....	2-130
5.17.2	Corrective Action System.....	2-131
5.18	Joint Technical and Management Reviews.....	2-132
5.18.1	Joint Technical Reviews.....	2-133
5.18.2	Joint Management Reviews.....	2-134
5.19	Software Risk Management.....	2-136
5.20	Software Management Indicators.....	2-138
5.20.1	Principal Objectives of Measurement.....	2-138
5.20.2	Continuous Improvement.....	2-139
5.20.3	Approach to Management Measurements.....	2-139
5.20.4	Key Software Management Questions.....	2-141
5.20.5	Software Measurement Set.....	2-141
5.20.6	Software Measurement Construct.....	2-141
5.20.7	Analysis and Reporting of Software Management Indicators.....	2-145
5.20.8	Software Indicator Thresholds and Red Flags.....	2-146
5.21	Security and Privacy Protection.....	2-146
5.22	Subcontractor Management.....	2-147
5.23	Interfacing with Software IV&V Agents.....	2-148
5.24	Coordination With Associate Developers.....	2-149
5.25	Improvement of Project Processes.....	2-149
5.25.1	Software Engineering Process Group.....	2-150
5.25.2	Process Audits.....	2-151
5.25.3	Change Implementation.....	2-152
5.25.4	SEPG Infrastructure.....	2-152
5.25.5	Process Training.....	2-152
5.25.6	Software Process Engineer/Lead.....	2-153
5.26	Software Sustainment (Optional).....	2-153
5.26.1	Software Sustainment Objectives.....	2-153
5.26.2	Planning for Software Sustainment.....	2-154
5.26.3	Software Maintenance Plan.....	2-155
5.26.4	The Software Sustainment Organization.....	2-156
5.26.5	Key Software Sustainment Issues.....	2-156
6.	Schedules and Activity Network.....	2-159
7.	Project Organization and Resources.....	2-161

7.1	Project Organization .....	2-161
7.2	Project Resources .....	2-162
7.2.1	Personnel Resources .....	2-162
7.2.2	Development Facilities .....	2-165
7.2.3	Government Furnished Equipment, Software and Services .....	2-165
7.2.4	Other Required Resources .....	2-165
7.2.5	Software Training Plans (Optional).....	2-166
8.	Notes .....	2-167
<b>Part 3. Additional SDP Guidebook Information .....</b>		<b>2-1</b>
AGI-1.	Software Roles and Responsibilities .....	3-3
AGI-2.	Bibliography .....	3-15
AGI-3.	Software-Related Definitions.....	3-17
AGI-4.	Software Acronyms.....	3-19
AGI-5.	Subject Index to the SDP Guidebook .....	3-21



## Figures

### Part 1. SDP Guidebook Introduction

Figure 1-1.	Organization of This SDP Guidebook.....	1-2
Figure 1-2.	Components of a Typical SDP Package—Example.....	1-5

### Part 2. Recommended Contents of a Program-Level SDP for Large Software Development Efforts

Figure 1.1.	Software Organization and Software Item Structure Overview—Example .....	2-2
Figure 1.2.1.	XMPL System Overview—Example .....	2-3
Figure 1.2.2.	XMPL Software System Architecture Overview—Example.....	2-4
Figure 1.3.3.	XMPL SDP Update Plan—Example.....	2-8
Figure 1.4.	Relationship Between the XMPL SDP and Other Key Plans—Example.....	2-9
Figure 3.1.	XMPL System Acquisition Lifecycle Phases—Example.....	2-13
Figure 3.2.	Software Process Levels Used In This Guidebook .....	2-14
Figure 3.4.	Software Lifecycle Development Domains—Example.....	2-15
Figure 3.5-1.	XMPL Software Development Process Overview—Example.....	2-16
Figure 3.5-2.	Principal Software Development Process Activities—Example.....	2-17
Figure 3.7.4.	Software Testing and Integration Process—Example .....	2-22
Figure 4.1.1.	Mission Critical Software Development Process—Example.....	2-25
Figure 4.1.2.	Support Software Development Process—Example .....	2-27
Figure 4.2.2.	Hierarchical Software Product Levels—Example.....	2-29
Figure 4.2.4.1.	COTS/Reuse Management Process—Example .....	2-31
Figure 5.1.1.	SDP Waiver Approval Process—Example .....	2-45
Figure 5.1.1.4.	Software Management from a Measurement Perspective—Example .....	2-49
Figure 5.2.3.	Electronic SDL Logical Partitioning—Example.....	2-54
Figure 5.4.	System/Segment Design Process Flow—Example .....	2-59
Figure 5.5-1.	The Origin of Software Requirements.....	2-63
Figure 5.6.2.	Software Item Architectural Design Process Flow—Example .....	2-72
Figure 5.6.3.	Software Item Detailed Design Process Flow—Example .....	2-74
Figure 5.7.	Software Coding and Unit Testing Process Flow—Example .....	2-80
Figure 5.8.	Software UI&T Process Flow—Example.....	2-85
Figure 5.9.	SIQT Process Flow—Example .....	2-92
Figure 5.10.	Hardware/Software Item Integration and Test Process—Example .....	2-100
Figure 5.14.	Relationship of the SDLs to the MSDL—Example .....	2-117
Figure 5.14.2.2.	Relationship of the Configuration Control Boards--Example.....	2-119
Figure 5.15.	Software Peer Review Process Overview—Example .....	2-123
Figure 5.16.1.	SQA Staffing Projection—Example.....	2-129
Figure 5.16.3.	SQA Independent Reporting Structure—Example.....	2-129
Figure 5.17.2.	Corrective Action Process Overview—Example .....	2-131
Figure 5.19.	Risk Management Process Overview—Example.....	2-136
Figure 5.20.2.	Closed Loop Software Control Process—Example.....	2-139
Figure 5.20.3.	Software Measurement Framework—Example .....	2-140
Figure 5.20.4.	Categories and Indicators Support the Key Management Questions—Example ..	2-141
Figure 5.20.6.	Elements of the Software Measurement Construct—Example.....	2-143
Figure 5.25.	Software Process Improvement Process Overview—Example.....	2-150
Figure 5.25.4.	SEPG Infrastructure—Example .....	2-152
Figure 7.1.	Overall Program Organization—Example.....	2-161
Figure 7.2.1.2.	Estimated Software Staff-Loading—Example.....	2-164

## Tables

### Part 1. SDP Guidebook Introduction

Table 1-1.	Common Acronyms Used in this Guidebook .....	1-3
------------	--	-----

### Part 2. Recommended Contents of a Program-Level SDP for Large Software Development Efforts

Table 1.2.3.1.	Mission Critical Software Class and SI Categories—Example .....	2-5
Table 1.2.3.2.	Support Software Class and SI Categories—Example .....	2-5
Table 1.2.3.3.	COTS/Reuse Software Class and SI Categories—Example .....	2-6
Table 3.3.	XMPL Software Items and Team Responsibilities—Example .....	2-14
Table 3.6.	XMPL Software Documentation Production Matrix—Example .....	2-18
Table 3.7.2.	Software Integration, Testing, and Verification Stages—Example .....	2-19
Table 3.7.3.	Software Integration, Testing and Verification Objectives—Example .....	2-20
Table 4.1.	Overview of Software Development Process Models—Example .....	2-24
Table 4.2.3.	Traceability Requirements by SI Category—Example .....	2-29
Table 4.2.10.1.	Candidate Software Management and Quality Control Plans—Example .....	2-39
Table 5.	Contents of SDP Section 5 .....	2-43
Table 5.1.	Readiness Criteria: Project Planning and Oversight—Example .....	2-44
Table 5.1.1.1.	Software Planning Tasks—Example .....	2-46
Table 5.1.1.3.	SI Build Delivery Plan-Example .....	2-47
Table 5.1.2.	Readiness Criteria: Software Test Plan—Example .....	2-50
Table 5.2.1-1.	Program-wide SEE CASE Tools—Example .....	2-52
Table 5.2.1-2.	SEE Development Sites—Example .....	2-52
Table 5.2.4.	Electronic SDF Organization—Example .....	2-55
Table 5.3.	Readiness Criteria: System/Segment Requirements Analysis—Example .....	2-57
Table 5.4-1.	Readiness Criteria: System/Segment Design—Example .....	2-58
Table 5.4-2.	System/Segment Design Tasks—Example .....	2-60
Table 5.5-1.	Readiness Criteria: Software Requirements Analysis—Example .....	2-64
Table 5.5-2.	Software Requirements Analysis Work Products—Example .....	2-64
Table 5.6-2.	Required Software Design Activity Work Products—Example .....	2-70
Table 5.6-3.	Roles and Responsibilities During Software Design—Example .....	2-70
Table 5.6.2.	Software Item Architectural Design Tasks—Example .....	2-73
Table 5.6.3.	Software Item Detailed Design Tasks .....	2-75
Table 5.7-1.	Readiness Criteria: Software Coding and Unit Testing—Example .....	2-78
Table 5.7-2.	Required Software Coding and Unit Testing Work Products—Example .....	2-78
Table 5.7-3.	Roles and Responsibilities During Software Coding and Unit Testing—Example .....	2-79
Table 5.7-4.	Software Coding and Unit Testing Tasks—Example .....	2-80
Table 5.8-1.	Readiness Criteria: Software Unit Integration and Testing—Example .....	2-84
Table 5.8-2.	Software UI&T Work Products—Example .....	2-84
Table 5.8-3.	Software UI&T Responsibilities—Example .....	2-85
Table 5.8-4.	Software UI&T Tasks—Example .....	2-86
Table 5.9-1.	Readiness Criteria: Software Item Qualification Testing—Example .....	2-89
Table 5.9-2.	Software Item Qualification Testing Work Products Per Build—Example .....	2-90
Table 5.9-3.	SIQT Roles and Responsibilities—Example .....	2-91
Table 5.9.3.	SIQT Preparation Tasks—Example .....	2-94
Table 5.9.4.	SIQT Dry Run Tasks—Example .....	2-95
Table 5.9.5.	Perform Formal SIQT Tasks—Example .....	2-96
Table 5.9.7.	Analyzing and Recording SIQT Results—Example .....	2-97



Table 5.10.	Readiness Criteria: Software/Hardware Item Integration and Testing—Example..	2-99
Table 5.10.1.	SI/HI Integration and Testing Preparation Tasks—Example.....	2-101
Table 5.10.2.	Performing SI/HI Integration and Testing Tasks—Example.....	2-102
Table 5.10.3.	Analyzing and Recording SI/HI Integration and Test Tasks—Example.....	2-103
Table 5.10.4.	Revision and Retesting SI/HI Integration and Test Tasks —Example.....	2-104
Table 5.11.	Readiness Criteria: System/Segment Qualification Testing—Example.....	2-105
Table 5.14-1.	Division of SCM Responsibilities—Example .....	2-116
Table 5.14-2.	Software Library Levels and Controls—Example .....	2-117
Table 5.15.1.2.	Software Development Peer Reviews—Example.....	2-125
Table 5.18.1.	Software Product Reviews By Activity and Category—Example .....	2-134
Table 5.18.2.	Software Documentation Maturity Mapped to Reviews—Example .....	2-135
Table 5.20.5.	Software Measurement Set—Example.....	2-142
Table 5.20.6-1.	Format of the Measurement Information Specification—Example .....	2-143
Table 5.20.6-2.	Example of a Measurement Information Specification for Staff Profile .....	2-144
Table 5.20.6-3.	Base and Derived Measure Specifications—Example .....	2-144
Table 5.20.6-4.	Format for the Measurement Indicator Specification—Example.....	2-145
Table 5.20.8-1.	Software Indicator Thresholds—Example.....	2-146
Table 5.20.8-2.	Software Indicator Program Red Flags—Example .....	2-146
Table 5.22.	Subcontractor Management Team Members and Responsibilities—Example....	2-148
Table 5.23.	Software IV&V Evaluations—Example.....	2-149
Table 5.25.1.	SEPG Membership and Responsibilities—Example.....	2-150
Table 5.25.2.	Focus of the Process Improvement Initiative—Example .....	2-151
Table 5.25.6.	Typical SPE Functions—Example.....	2-153
Table 5.26.3.	Example Outline of the Software Maintenance Plan.....	2-156
Table 5.26.5.	Key Software Sustainment Issues .....	2-157
Table 7.2.1.	Chief Software Engineer Team Responsibilities—Example .....	2-163
Table 7.2.1.3.	Estimated Skill Levels By Location and Function—Example.....	2-164
Table 7.2.2-1.	Team Locations and Software Activities—Example .....	2-165
Table 7.2.2-2.	Facilities Allocation—Example .....	2-165
Table 8.1.	Acronyms—Example .....	2-167
Table 8.2.	Software-Related Definitions—Example .....	2-167
Table 8.3.	Work Instructions and Procedures—Example .....	2-168

### Part 3. Additional SDP Guidebook Information

Table AGI-1.	Roles and Responsibilities of the Chief Software Engineer—Example .....	3-4
Table AGI-3.	Roles and Responsibilities of the Software Process Lead—Example .....	3-6
Table AGI-4.	Roles and Responsibilities of the IPT Software Lead—Example.....	3-7
Table AGI-5.	Roles and Responsibilities of the IPT Software Integration and Test Lead— Example.....	3-8
Table AGI-6.	Roles and Responsibilities of the Software Item Lead—Example.....	3-9
Table AGI-7.	Roles and Responsibilities of the Software Engineer—Example .....	3-10
Table AGI-8.	Roles and Responsibilities of the Software Test Engineer—Example.....	3-11
Table AGI-9.	Roles and Responsibilities of the Software Configuration Management— Example.....	3-12
Table AGI-10.	Roles and Responsibilities of the Software Quality Assurance Management— Example.....	3-13
Table AGI-11.	Roles and Responsibilities of the Software Subcontract Management—Example.	3-14

## Part 1. SDP Guidebook Introduction

### 1. Scope and Perspective

The contents and organization of the Software Development Plan (SDP) recommended in this Guidebook is based on guidelines as defined in:

- Section E.2.1 of the “EIA/IEEE Interim Standard J-STD-016-1995” (hereafter referred to as **J-16**),
- Department of Defense (DoD) Data Item Description (DID) DI-IPSC-81427A, Software Development Plan, and
- The Aerospace Corporation software development standard Technical Operating Report, TOR-2004(3909)-3537B, “*Software Development Standard for Space Systems*” (hereafter referred to as **TOR-3537B**). Appendix H of TOR-3537B contains the SDP content template.

This Guidebook is compliant with those standards, however, **TOR-3537B is the cited standard** as it is newer (published 11 March 2005) and is currently being used as the compliance standard on United States Air Force (USAF) Space and Missile Systems Center (SMC) programs. TOR-3537B has also been published as **SMC Standard SMC-S-012, “Software Development for Space Systems”** dated 13 June 2008. If the guidance being applied appears only in J-16, then J-16 is the cited standard. There is no intent to duplicate the information contained in those standards. **The intent of this Guidebook is to supplement the standards with detailed guidance, recommend contents, and examples, to assist in the preparation and review of SDPs.** Therefore, this Guidebook should be used in conjunction with the standards.

The contents of a SDP, as defined collectively by the above standards, consists of the following eight sections plus Addendums and Annexes as needed:

1. Scope
  2. Referenced Documents
  3. Overview of Required Work
  4. General Requirements
  5. Detailed Requirements
  6. Schedules and Activity Network
  7. Project Organization and Resources
  8. Notes
- Addendums
  - Annexes

### 2. Organization of this SDP Guidebook

This Guidebook is organized into three parts as shown in Figure I-1. **Part 1**, the introduction, covers the basic approach and general information of special importance to the reader. **Part 2** of this SDP Guidebook constitutes the bulk of the document as it contains the recommended contents of a program-level SDP in terms of what is expected and recommended to be included within each subsection or paragraph and examples of expected contents, figures, and tables. The Notes section (Section 8) contains acronyms, definition of terms, and an example list of work instructions that document how to carry out tasks described in the SDP. **Part 3**, Additional Guidebook Information, contains a suggested list of software roles and responsibilities, references, definitions, acronyms, and a Subject Index to this Guidebook.





- **PART 1: SDP Guidebook Introduction**
- **PART 2: Recommended Contents of a Program-Level SDP for Large Software Development Efforts**
- **PART 3: Additional Guidebook Information**

Figure 1-1. Organization of This SDP Guidebook

### 3. Electronic Data Management

This Guidebook is written with the assumption that the contractor's parent organization has in place an effective and comprehensive Electronic Data Interchange Network (EDIN) for the storage, retrieval and distribution of program related software documentation and work products (see subparagraph 5.2.3.1).

### 4. Example Text and Highlights

**Tailoring.** Throughout this Guidebook the name of a fictitious example program will be called "XMPL." All of the figures and tables used in this SDP Guidebook are examples and they are expected to be tailored for each program's SDP and be compliant with the developer's Standard Software Process (SSP).

**Example Text.** In some sections of this Guidebook, example text is included as a guide for preparation of that section. Example text is identified as follows:

***Example Text:***

The example text provided in this Guidebook is outlined with a solid outside border and includes the words "*Example Text*" in the upper left corner.

**Highlights.** Paragraphs or sentences containing essential or key information are highlighted with a light yellow background. When the term "<corporate>" is used in example text, the intention is to replace it with the name of the parent organization of the program producing the SDP.

### 5. Terms and Acronyms Used

Terms used in this Guidebook are consistent with the definitions in Section 3 of TOR-3537B.

This Guidebook is not a standard! Therefore, there are no mandatory "shalls." Instead, the following terms—and what they mean—are used throughout this Guidebook:

- **Must: Highly recommended for compliance with TOR-3537B and J-16. The word "must" is in bold letters to highlight that it is, or is implicitly, a "shall" in the standards.**
- **Should: Recommended for completeness**
- **Can: Discretionary but should be seriously considered for inclusion**
- **May: Discretionary or used to show examples**

Using the term "may" implies that other good options exist—choosing between them is left up to the program.

**Acronyms.** Acronyms are used extensively in this Guidebook. Acronyms and definition of terms used are included in Part 3 of this Guidebook. Table 1-1 is a list of the most common acronyms used throughout the Guidebook. It is expected that individual sections of this Guidebook will likely be used as a reference when needed (as opposed to assuming the reader will always read the entire Guidebook). Consequently, acronyms are typically redefined when first encountered in each section.

Table 1-1. Common Acronyms Used in this Guidebook

<b>CSWE</b>	Chief Software Engineer	<b>SDF</b>	Software Development File (or Folder)
<b>CCB</b>	Configuration Control Board	<b>SDL</b>	Software Development Library
<b>CDRL</b>	Contract Data Requirements List	<b>SDP</b>	Software Development Plan
<b>COTS</b>	Commercial Off-The-Shelf	<b>SDR</b>	Software Discrepancy Report
<b>C/R</b>	COTS/Reuse (a software class)	<b>SEIT</b>	System Engineering, Integration, and Test
<b>CUT</b>	Code and Unit Test	<b>SEPG</b>	Software Engineering Process Group
<b>IPT</b>	Integrated Product Team	<b>SI</b>	Software Item
<b>IMP</b>	Integrated Master Plan	<b>SPR</b>	Software Peer Review
<b>IMS</b>	Integrated Master Schedule	<b>SQA</b>	Software Quality Assurance
<b>MC</b>	Mission Critical (a software class)	<b>SS</b>	Support Software (a software class)
<b>MSDL</b>	Master SDL	<b>SU</b>	Software Unit
<b>SCM</b>	Software Configuration Management	<b>SW/CCB</b>	Software Configuration Control Board
<b>SCR</b>	Software Change Request/Report	<b>TIM</b>	Technical Interchange Meeting

The “Program Office” or the “Acquisition Program Office” and the “customer,” as referenced in this Guidebook, refers to the government organization responsible for the program’s contract and implicitly includes their representatives—such as personnel from The Aerospace Corporation, other Federally Funded Research and Development Centers (FFRDCs), and System Engineering and Technical Assistance (SETA) contractors.

## 6. Format of the Process Descriptions

A graphical and tabular emphasis is heavily displayed in this SDP Guidebook and is the recommended format to more clearly describe the software development processes. Details of the software development process are contained in subsections of SDP Section 5, especially subsections 5.3 through 5.11, covering the principal activities of the software development process.

The following four inter-related items (three tables and a flowchart) are recommended for inclusion in SDP subsections, 5.3 through 5.11, to provide a comprehensive definition of the software tasks involved in each activity:

- **Readiness Criteria Table:** Should contain: Entry Criteria; Exit Criteria; Verification Criteria; and Measurements for each software development activity
- **Software Work Products Table:** Should contain: A list of work products required, or typically produced, for each software development activity organized by software category
- **Input/Process/Output (IPO) Flowchart:** Should show: the input documents and work products, process tasks, and outputs for each software development activity
- **Task Table:** Must be linked to the process activities in the IPO flowchart but containing more details of the tasks and sub-tasks for each software development activity. The IPO flowchart can be considered optional, but the Task Tables should be included in subsections 5.3 through 5.11.



Examples of these tables, and the flowchart, are included in subsections 5.3 through 5.11 of this Guidebook. Example figures throughout this Guidebook are intentionally made simple to convey the general content expected in the figure. In most cases, it is expected that the figures produced by the contractor for their SDP will have more content and detail than the examples shown.

## 7. Integrated Product Teams (IPTs)

The establishment of effective software IPTs is one of the most important ingredients to a successful software development program. The software IPTs, referenced extensively throughout this Guidebook, **must** be composed of relevant stakeholders who make and implement decisions for the work being developed. The software IPTs are collectively responsible for delivering the product(s) and its members should:

- Share a common understanding of the IPTs tasks, objectives, and responsibilities
- Collectively provide the skills and expertise needed to accomplish the tasks and objectives
- Collaborate internally and externally with other IPTs and relevant stakeholders
- Provide the advocacy and representation to address all phases of the lifecycle

## 8. Analysis and Design Methodologies

The recommendations in this Guidebook are applicable to all software analysis and design methodologies; however, the examples presented in this Guidebook assume the software is being developed using an Object-Oriented approach since Object-Oriented Analysis (OOA) and Object-Oriented Design (OOD) have, to a large extent, replaced the Structured Analysis (SA) and Structured Design (SD) approach commonly used for the past 30 years. The scope of this SDP Guidebook does not permit a discussion and evaluation of the advantages and disadvantages of various methodologies. Newer methodologies, such as Agile and Extreme Programming, may be appropriate for some types of software development.

## 9. Format Options for the SDP

A comprehensive SDP is composed of multiple parts. Typically, there are two basic approaches to SDP formats: programs with a single SDP and programs with a program-level SDP plus site-specific SDPs.

**The Single SDP Approach.** A program may elect to have a single SDP and mandate that it be followed by all software team members. That approach works very well when all developers, including subcontractors, are co-located and using the prime's infrastructure.

**The Site-Specific SDP Approach.** On large programs, typically involving numerous corporations that are geographically dispersed, site-specific SDPs are often needed because of significant corporate differences in software organization, management policies, development environments, and unique operational processes and procedures. Site-specific SDPs are written and maintained by the development sites and provide additional standards and procedures specific to each site. They expand upon, but **must not** conflict with, the processes and procedures defined in the program-level SDP unless a waiver has been approved. Figure 1-2 is a typical organization of the complete "SDP package" containing three parts including site-specific annexes. Programs with a single SDP would not have Part 3.

**SDP Plans.** All of the plans listed as SDP Addendums in Figure 1-2 are recommended as long as they are applicable. Some programs will require the plans listed as SDP addendums embedded in the SDP itself; other programs may require them to be separate documents. Software management and quality control plans are briefly described in subparagraph 4.2.10.1 of this Guidebook.

PART 1	Program-Level Software Development Plan	
	Appendices	
PART 2 SDP Addendums	<b>SDP Management Plans</b> <ul style="list-style-type: none"> <li>• Software Quantitative Management Plan</li> <li>• Software Metrics Plan</li> <li>• Software Subcontract Management Plan</li> <li>• Software Risk Management Plan</li> <li>• Software Reviews Plan</li> <li>• Software COTS/Reuse Plan</li> <li>• Software Resource Estimation Plan</li> <li>• Software Integration and Test Plan</li> <li>• Software Maintenance Plan</li> </ul>	
	<b>SDP Quality Control Plans</b> <ul style="list-style-type: none"> <li>• Software Configuration Management Plan</li> <li>• Software Quality Assurance Plan</li> <li>• Software Process Improvement Plan</li> <li>• Software Corrective Action Plan</li> <li>• Software Product Inspection Plan</li> <li>• Software Standards (Coding, Design, etc.)</li> </ul>	
PART 3 Site-Specific SDPs	Annex A: Site 1 Specific SDP Annex B: Site 2 Specific SDP Annex C: Site 3 Specific SDP	

Figure 1-2. Components of a Typical SDP Package—Example

## 10. Tailoring of the SDP

The SDP **must** be tailored to the specific requirements of a particular program, program phase, or contractual structure to which it applies. Although tailoring is generally a responsibility of the acquirer, prospective and selected software developers may provide suggested tailoring. Generic tailoring guidance is provided in J-16 Annexes A, B, and C. Tasks that add unnecessary costs, and data that does not add value to the product, **must** be eliminated. Tailoring can include deletion, alteration, or addition of activities as long as the result satisfies program requirements. Acquirer-generated tailoring is normally specified in the Statement of Work (SOW), Compliance Documents or in the Contract Data Requirement List (CDRL) section of the contract.

## 11. Large Versus Small Software Developments

SDP tailoring guidelines apply to both large and small development efforts. If a specified task or activity does not make sense because of the size of the development effort, it should be deleted. There is no intention to shoot a mouse with an elephant gun. However, a sound software process management philosophy dictates that all software developments (large and small) go through the same procedural steps—the difference is a matter of scale.

## 12. Recommended SDP Numbering Format

To enhance readability, it is recommended that the SDP numbering format does not go beyond four levels plus two additional unnumbered levels as follows:

- **Level 1:** Section (Example 5)
- **Level 2:** Subsection (Example 5.1)
- **Level 3:** Paragraph (Example 5.1.1)
- **Level 4:** Subparagraph (Example 5.1.1.1)
- **Level 5:** Bold key word(s) to lead off the paragraph
- **Level 6:** Bullets indented under Level 5 (Note: Bullets can also be used at Levels 2 through 4)



## Part 2. Recommended Contents of a Program-Level SDP for Large Software Development Efforts

### 1. Scope

The SDP starts with the Scope and is defined by TOR-3537B<sup>1</sup> as containing four subsections: Identification (1.1), System Overview (1.2), Document Overview (1.3), and Relationship to Other Plans (1.4).

#### 1.1 Identification

The purpose of this subsection is to fully identify the system, the software to be produced, and the activities to which the SDP applies. It includes applicable identification numbers, version numbers, and release numbers. Subsection 1.1 can be as short as one paragraph or a half page or longer to introduce the SDP and organization of the Software Item (SI)<sup>2</sup>. For example, an introduction to the SDP may be similar to the following:

##### *Example Text:*

This Software Development Plan (SDP) establishes the management and technical plans to be used during Phase-C, Complete Design, by the XMPL Integrated Product Teams (IPTs), in the development of software items for all segments and their development sites.

This SDP describes the organization, processes, controls, and tools applied to the management, design, development, and test of the XMPL software products. This plan applies to all software integrated into XMPL during its lifecycle, including newly developed software, reused software and modifications to it, and commercial off-the-shelf (COTS) products.

The SDP provides software management with the controls necessary to oversee the XMPL software development activities. It provides software engineers with the standards and practices required for all XMPL software development. This SDP implements the <corporate> Standard Software Process (SSP), as tailored for the XMPL program.

Subsection 1.1 should contain a software organization overview as shown in the example Figure 1.1. This figure should show the program segments containing software, the Software Items (SIs), and a top-level view of the software organization. A description of the software organization **must** also be addressed in subsection 7.1 of the SDP. Unfortunately, subsection 7.1 of TOR-3537B and J-16 is titled "Project Organization" and many SDP authors take that literally to mean "project" and do not show details of the software organization. In the context of an SDP, subsection 7.1 **must be interpreted to mean a view of the software organization from a project perspective.**

Some programs may not have all the software titles shown in Figure 1.1. In that event, responsibilities identified for the Chief Software Engineer (CSWE), Chief Software Architect, and Chief Process Engineer should be performed by the person(s) having those responsibilities regardless of their job title. This Guidebook assumes the program has a CSWE and contains descriptions of the responsibilities typically performed by the CSWE (see AGI-1 Tables AGI-1 and AGI-2 and subparagraph 7.2.1.1).

<sup>1</sup> TOR-3537B is cited throughout this Guidebook, however, J-16 can also be used as the referenced standard since this Guidebook is compliant with both standards.

<sup>2</sup> The SI was called a Computer Software Configuration Item (CSCI) in MILSTD-2167A and MILSTD-498.

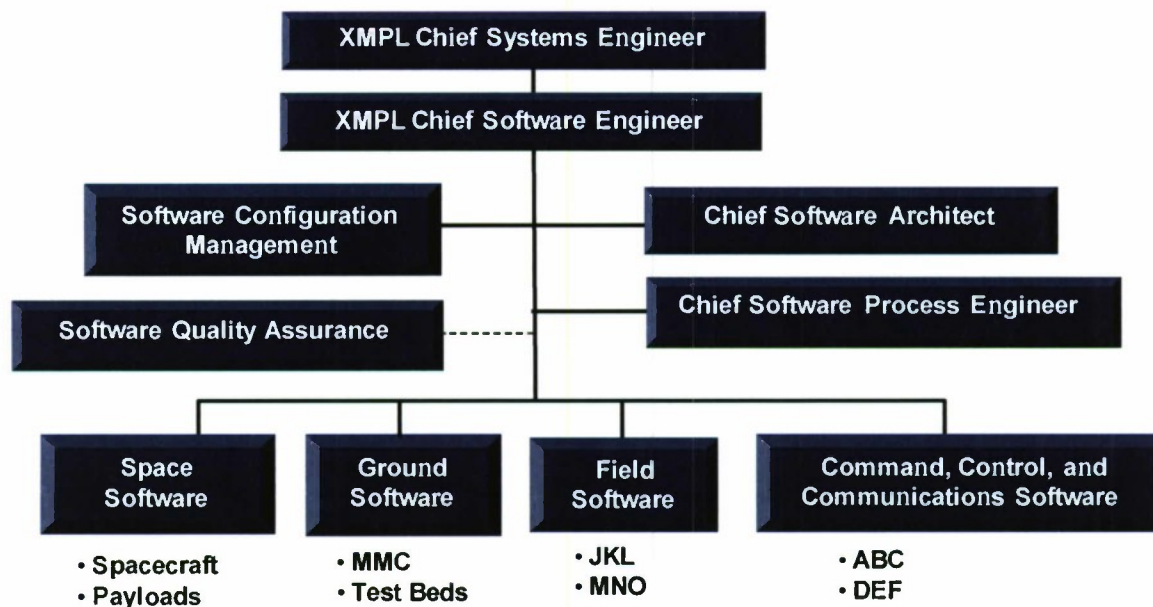


Figure 1.1. Software Organization and Software Item Structure Overview—Example

## 1.2 System Overview

The intent of this subsection is to describe the general nature of the system and the software. To provide a clear overview of the “system” versus the “software,” it is recommended that subsection 1.2 be broken into two paragraphs: System Architecture Overview (1.2.1) and Software Architecture Overview (paragraph 1.2.2). Paragraph 1.2.1 should be further broken down into a general system description followed by short descriptions of the segments comprising the overall system.

### 1.2.1 System Architecture Overview

The purpose of the system **must** be briefly stated in paragraph 1.2.1. As applicable, it **must** summarize any historical aspects of the system to be developed and identify the project sponsor, acquirer, user(s), developers, as well as planned maintenance organizations and operating sites. The segments that comprise the system **must** be listed and an overall graphical diagram of the system should be included similar to the example shown in Figure 1.2-1.

The remainder of SDP paragraph 1.2.1 should contain as many single paragraphs as necessary to describe the segments that involve software responsibilities for the system. In the XMPL example there would be descriptions of the following four segments:

- **Space Segment:** Top-level functions of the spacecraft and payload software
- **Command, Control, and Communications (C3) Segment:** Top-level functions of C3 software
- **Ground Segment:** Top-level functions of the ground-based software
- **Field Segment:** Top-level functions of the field software



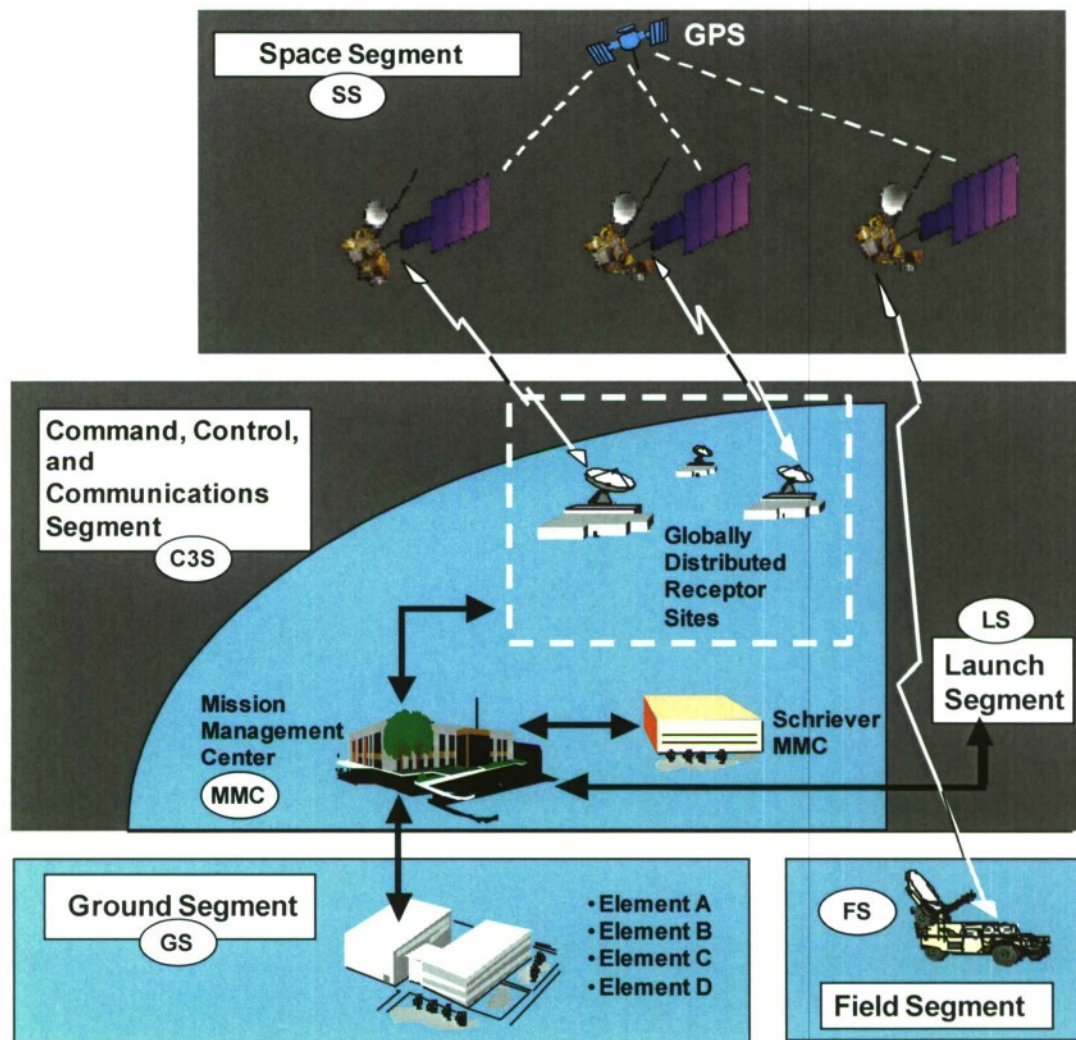


Figure 1.2.1. XMPL System Overview—Example

### 1.2.2 Software Architecture Overview

This paragraph provides an overview of the software system (or functional) architecture, a definition of the software categories, and an overview of the Software Items (SI) and responsibilities.

The overall software system architecture should be depicted in a diagram; Figure 1.2.2 is an example of such a diagram. An additional, or optional approach, would be to include a “functional matrix” table showing the software “functionality” for each segment or SI. A physical overview of the system may also be necessary.



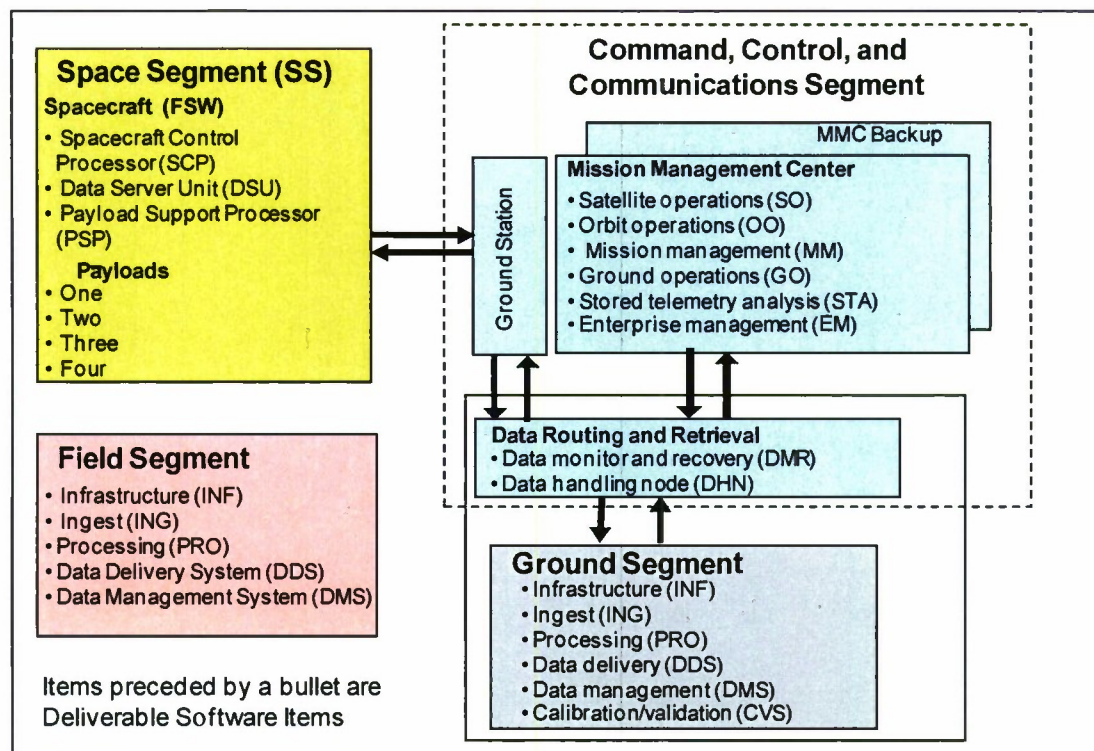


Figure 1.2.2. XMPL Software System Architecture Overview—Example

### 1.2.3 Software Classes and Categories

There are typically three generic classes of software in a software-intensive system: mission critical software, support software, and COTS/Reuse software as described in example Tables 1.2.3.1 through 1.2.3.3. Each software class can be further sub-divided into categories as needed for the program, resulting in the identification of 4–8 categories of software for a typical program.

The number of software classes, the number of categories within those classes, and the names of each are **not critical**. What is important is that there **must** be a definition of the category assigned to each software entity because not every software entity needs to have the full set of documentation, the full set of reviews, the full set of metrics, and the same level of testing.

Assigning categories to software entities can result in cost savings by eliminating unnecessary documents, reviews, metrics, and testing. However, the simplicity of this approach is deceiving since obtaining agreements from all stakeholders on the appropriate category to assign is not always simple.

#### 1.2.3.1 Mission Critical Software

Mission Critical (MC) software is physically part of, dedicated to, and/or essential to the mission performance of the system. It includes both space and ground software. MC software may be expanded to two software categories as defined by the example in Table 1.2.3.1.

Table 1.2.3.1. Mission Critical Software Class and SI Categories—Example

Class Definition	Category	Category Definition
<b>MC</b>	<b>MC-1</b>	Deliverable applications software that plays a direct role in system operation and system development.
<b>MISSION CRITICAL SOFTWARE</b> Applications software used to perform real time operations and non-real time functions implicitly required for a mission.	<b>MC-2</b>	Same as MC-1 but the software is embedded in deliverable hardware. Firmware is software and is treated in the same way as software that executes in general purpose computers.

### 1.2.3.2 Support Software

Support Software (SS) aids in system hardware and software development, test, integration, qualification, and maintenance. The SS class may be composed of three SI categories, SS-1, SS-2, and SS-3 as defined in Table 1.2.3.2. MC-1, MC-2, and SS-1 software categories (but not SS-2 or SS-3) are usually deliverable and contractually obligated, **must** pass through **all** of the developmental phases, including all of the relevant software documentation, reviews, metrics, and testing, and are subject to external Software Discrepancy Reports (SDRs).

SS-2 software is used in non-operational environments, may be deliverable, but normally not contractually obligated. Both SS-2 and SS-3 software categories do not go through the full software lifecycle or receive external SDRs and are normally not deliverable. However, in some cases, important support software may be contractually deliverable. For example, deliverable support software may include training software, database-related software, software used in automatic test equipment, and simulation software used for diagnostic purposes during the maintenance activity. The contractor **must** decide the appropriate category for all software entities in compliance with contractual requirements.

Table 1.2.3.2. Support Software Class and SI Categories—Example

Class Definition	Category	Category Definition
<b>SS</b>	<b>SS-1</b>	Software items that play a direct role in program and system development including software and system requirements qualification and acceptance testing for final "sell-off."
<b>SUPPORT SOFTWARE</b> Software that aids in system hardware and software development, test, integration, qualification and maintenance.	<b>SS-2</b>	Support software that is typically prototype software, simulation software, or performance analysis and modeling tools (although some of this type of software may be selected to be in category SS-1).
	<b>SS-3</b>	Non-deliverable and non-critical tools or test drivers that indirectly aid in the development of the other categories of software.

### 1.2.3.3 Commercial Off-The-Shelf and Reuse Software

COTS/Reuse software is non-developmental software items including commercial and government off-the-shelf (COTS or GOTS) software as well as reused software obtained from internal libraries, previously developed under an internal research and development effort, or developed by other programs, set up specifically for reuse. The C/R class may be composed of two categories as described by the example in Table 1.2.3.3.



Table 1.2.3.3 COTS/Reuse Software Class and SI Categories—Example

Class Definition	Category	Category Definition
<b>C/R</b>	<b>C/R-1</b>	Non-developmental software that is <b>unmodified</b> COTS or Reused software.
<b>COTS/REUSE SOFTWARE</b> Non-developmental software items including commercial and government off-the-shelf (COTS or GOTS) software. All C/R products <b>must</b> be treated and controlled as defined for the category targeted for its end use.	<b>C/R-2</b>	Non-developmental software that is <b>modified</b> COTS or Reused software.* (A distinction between vendor-provided and internally reused software may be made for C/R-1 and C/R-2 if meaningful to the program)

\*Modifying vendor-provided COTS is generally a high-risk approach and is **not recommended**.

**Calculating ESLOC.** When software design and/or code is reused, the costing of it is usually based on an approach called the “Equivalent Source Lines of Code” (ESLOC) count. The premise is that some portion of the design, code and/or testing does not have to be redone and can be reused. The method to be used for calculating ESLOC must be described in the SDP.

One common approach to calculating ESLOC is to set the proportionate weighting factors for designing, coding and testing the reused software product to 40%, 30%, and 30% respectively. Programs may deviate from these standard proportions (40%, 20%, and 40% is also often used). The ESLOC count is calculated by estimating the percentage of new design, coding and testing needed for the deliverable product, and multiplying the sum of these weightings by the lines of code in the reused product.

For example, assume an existing documented software product with 1000 source lines of code was selected for reuse by another program having a need for similar functionality. Upon examination of the reused product, an estimate is made that only 10% of the design needs to be changed, 30% of the code must be redone, and 60% of the software needs to be retested. In this example, the ESLOC is 310 and is calculated as follows:  $1000 [(0.1 * .4) + (.3 * .3) + (.6 * .3)] = 1000 [.04 + .09 + .18] = 1000 [.31] = 310$ .

#### 1.2.3.4 Software Category Features

A single Software Item (SI) may consist of different classes and/or categories. In that event, each part of the SI **must** be compliant with the documentation, review, and testing requirements of the category assigned to it. All software releases **must** be configuration controlled by a Software Development Library (SDL) at the segment level or by the Master Software Development Library (MSDL) at the program level as described in SDP paragraph 5.2.3.

Software cannot be moved up or “promoted” to a higher category level without additional development and testing. To achieve a higher category level, the software **must** be “re-engineered” and conform to the documentation, review, and testing requirements imposed on the higher category level. All COTS and reused products **must** be treated and controlled as defined for the category targeted for its end use.

### 1.3 Document Overview

This overview of the SDP document **must** include its constituent parts and organization, and should include a plan for updating. If applicable, it **must** also describe any security, distribution, or privacy protection considerations associated with its use.

#### 1.3.1 SDP Component Parts

The SDP is more than just a program-level document since it usually contains addendums and annexes that may be bound separately from the main volume. These SDP components can be shown in graphical form on the page following the title. The following is an example of text that may be used for paragraph 1.3.1:

***Example Text:***

The complete XMPL SDP is organized into three parts as follows:

**Part 1:** This is the program-level SDP (also called the SDP “main volume”)

**Part 2:** Addenda to the SDP containing XMPL plans or processes documents:

Addendum A: Software Metrics Plan

Addendum B: Software Roles and Responsibilities

Addendum C: Software Subcontractor Management Plan

Addendum D: Software Quality Assurance Plan

Addendum E: Software Configuration Management Plan

Addendum F: Software Reviews Plan

Addendum G: Software Resource Estimation Plan

Addendum H: Software COTS/Reuse Plan

Addendum I: Software Integration and Test Plan

Addendum J: Software Risk Mitigation Plan

Addendum K: Software Maintenance Plan

Addendum L: Software Training Plan

**Part 3:** Annexes to the SDP—Site-Specific SDPs as required for software team members

### 1.3.2 SDP Organization

This paragraph of the SDP is essentially “boiler-plate” as it describes the format required in the standard used to produce it—in this case, TOR-3537B. The following example text may be used for this paragraph:

**Example Text:**

This SDP was produced using the compliance standard entitled “*Technical Operating Report, TOR-2004(3909)-3537B, “Software Development Standard for Space Systems.”* The XMPL SDP is organized into the following eight sections:

- **Section 1:** Provides overviews of the XMPL system, the software system, SDP updates, software classes and categories, and the relationship of the XMPL SDP to other XMPL documents
- **Section 2:** Identifies all documents referenced by this SDP
- **Section 3:** Discusses an overview of the work to be performed. It describes the requirements and constraints on the software, documentation, schedules, and resources
- **Section 4:** Describes the general software development activities to be performed. This includes an overview of the software development process, standards that apply to the development activities, the approach to developing and incorporating reusable software, information on computer resource utilization, and the handling of critical requirements
- **Section 5:** Provides details on each of the individual software development phases and activities that are to be performed, or may be performed. It covers project planning, methods, and the tools that support these methods
- **Section 6:** Identifies the schedules and activities to be performed
- **Section 7:** Provides details on the XMPL project organization and the resources to be applied
- **Section 8:** Provides the definition of acronyms and selected terms used in this document plus identification of lower level standards and procedures

### 1.3.3 SDP Updates

The SDP is considered a “living” document that **must** be updated periodically throughout the software development lifecycle. Updates are usually planned to occur at the Program Milestones, and a figure similar to the example Figure 1.3.3 can be included in the SDP—or the same information provided in table format.

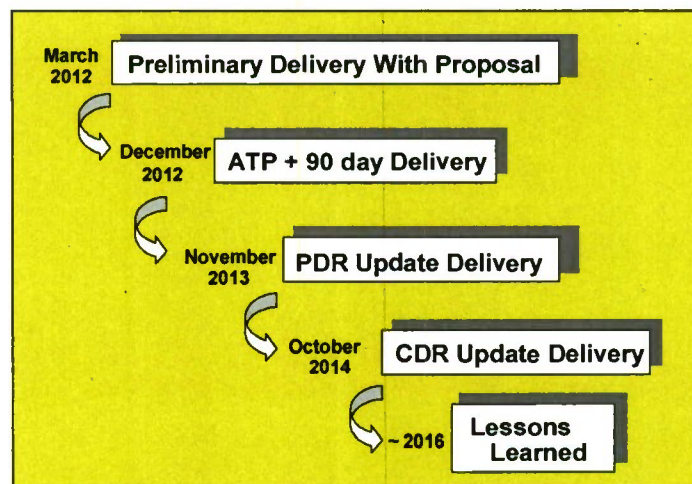


Figure 1.3.3. XMPL SDP Update Plan—Example



## 1.4 Relationship to Other Plans

The relationship of the SDP to other key project management plans is important to establish document subordination in the event of conflicts between plans. Figure 1.4 is an example overview of the relationship of the SDP to other key plans; software documents are highlighted. Example text for this subsection may be:

**Example Text:**

The XMPL SDP is compliant with the <corporate> Standard Software Process and serves as the compliance document for all XMPL software development. Contractor specific plans, development policies, and practices are incorporated as annexes to this program-level SDP.

Team members shall comply with this SDP based on tailoring guidance provided in subsection 4.1 and captured in their annexes to this document. The XMPL SDP is subordinate to the Integrated Master Plan (IMP) and, in the event of a conflict, the IMP takes precedence. The SDP is not subordinate to, but must be consistent with, the other plans at the same peer level as shown in Figure 1.4 (e.g., SEMP, CM, etc.).

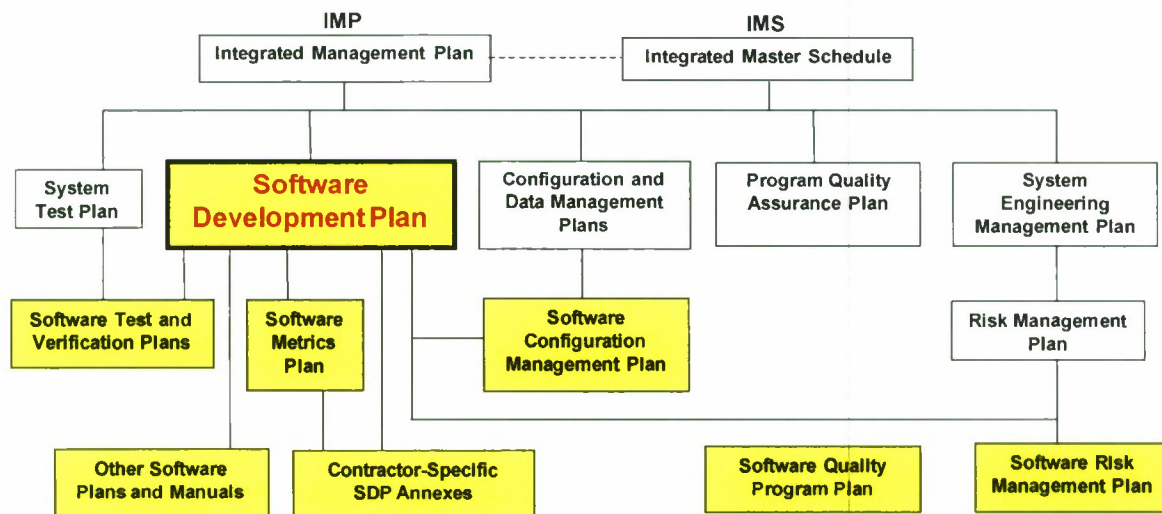


Figure 1.4. Relationship Between the XMPL SDP and Other Key Plans—Example



## 2. Referenced Documents

All referenced and applicable documents in the SDP **must** be listed in Section 2 and **must** contain the document number, document title, and date of the revision used. A tabular format is an easy way to display this information and should be organized by government and non-government documents and then broken down into referenced and applicable documents as shown in the examples below.

Referenced documents are guidelines, but Applicable documents **must be adhered to**. Non-Government Applicable documents are usually mandated by the developer's organization or by the program.

### 2.1 Government Documents

#### 2.1.1 Government Referenced Documents—Example

Document Number	Document Title	Revision Date
<i>Document Number</i>	Document Title	<i>Document Date</i>
<i>Document Number</i>	Document Title	<i>Document Date</i>

#### 2.1.2 Government Applicable Documents—Example

Document Number	Document Title	Revision Date
<i>Document Number</i>	Technical Requirements Document (TRD)	<i>Document Date</i>
<i>Document Number</i>	Interface Control Document (ICD)	<i>Document Date</i>

### 2.2 Non-Government Documents

#### 2.2.1 Non-Government Referenced Documents—Example

Document Number	Document Title	Revision Date
<i>Document Number</i>	Software Estimating Guide	<i>Document Date</i>
ISO 9001	Quality Program	<i>Document Date</i>
ISO/IEC 15939	Software Engineering—Software Measurement Process	2002
<i>Document Number</i>	Software Peer Review Guide	<i>Document Date</i>
IEEE-1471	Software Architecture Descriptions	<i>Document Date</i>
AIAA R-023A	Recommended Practice—Human Computer Interface for Space System Operations	1995

#### 2.2.2 Non-Government Applicable Documents—Example

Document Number	Document Title	Revision Date
Aerospace Report No. TOR-2004(3909)-3537B	Software Development Standard for Space Systems	11 March 2005
J-STD-016-1995	Standard for Information Technology	September 1995
ANSI/ISO/IEC 9899	C	1990
ISO/IEC 14882	C++	July 1998
<i>Document Number</i>	<Corporate> Standard Software Process	<i>Document Date</i>
<i>Document Number</i>	Software Subcontract Management Guidebook	<i>Document Date</i>
<i>Document Number</i>	Configuration and Data Management Plan	<i>Document Date</i>
<i>Document Number</i>	Risk Management Plan	<i>Document Date</i>
<i>Document Number</i>	Integrated Management Plan (IMP)	<i>Document Date</i>
<i>Document Number</i>	Integrated Management Schedule (IMS)	<i>Document Date</i>
<i>Document Number</i>	Security Implementation Plan	<i>Document Date</i>
<i>Document Number</i>	Integration and Test Plan	<i>Document Date</i>





### 3. Overview of Required Work

There are no specific numbered subsections required for Section 3 in TOR-3537B. However, TOR-3537B describes Section 3 as containing an overview of requirements and constraints on the: system, software, documentation, development strategy, schedule, resources, and other areas, such as contractual and non-contractual constraints, plus a requirement to show the position in the system lifecycle where the SDP applies. The following organization is recommended.

#### 3.1 System Acquisition Lifecycle

A figure similar to example Figure 3.1, or a table, should be included in SDP subsection 3.1 to provide a top-level overview of the system acquisition lifecycle phases combined with a clear indication as to where in the system lifecycle the SDP being written applies. Also, the program's Integrated Master Plan (IMP) **must** be referenced in the SDP since the IMP includes important information on program tasks, events, and milestones for software activities.

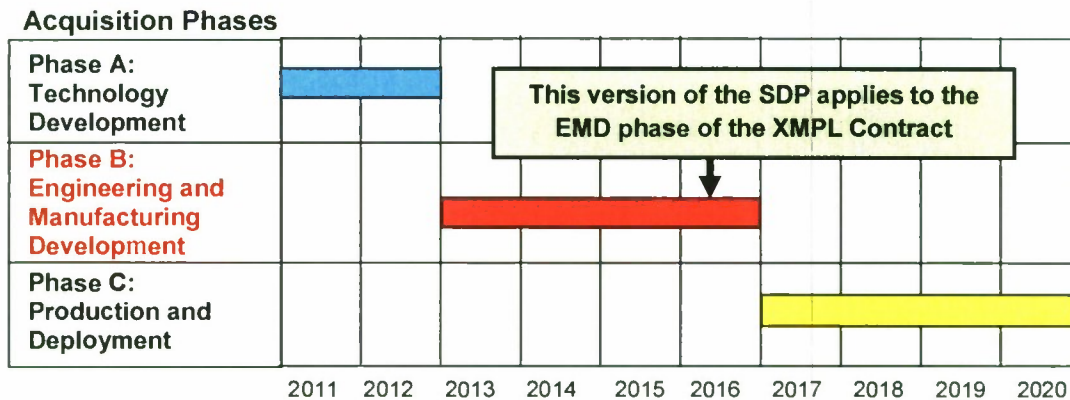


Figure 3.1. XMPL System Acquisition Lifecycle Phases—Example

#### 3.2 Software Requirements and Constraints

Figure 3.2 is a depiction of the basic levels of abstraction for describing the software process used in this Guidebook. The top level is focused on **programmatic phases**. The middle level incorporates the principal **software development activities** required by subsections 5.3 through 5.11 of the SDP. The lowest process level involves the **specific tasks** required to carry out the software development activities.

There are many types of system requirements and constraints that may become drivers for the software. Such drivers may include: specific standards that **must** be followed; precise performance mandates; requirements to execute on a government platform; preliminary deliveries of software such as an interim version needed to support military exercises; mandated severe schedule constraints to meet launch or delivery dates; etc. (see subsections 3.8 and 3.9).

A fundamental aspect of the software development process is the system lifecycle model to be followed. A detailed discussion of process models is beyond the scope of this document. However, six of the most common software development process models (Prototype, Waterfall, Incremental, Evolutionary, Spiral, and Compound—such as the Rational Unified Process™) are briefly defined in subsection 4.1. The process models are also discussed in the Air Force Space and Missile Systems Center Instruction 63-104, dated 21 November 2005, and in many other sources.

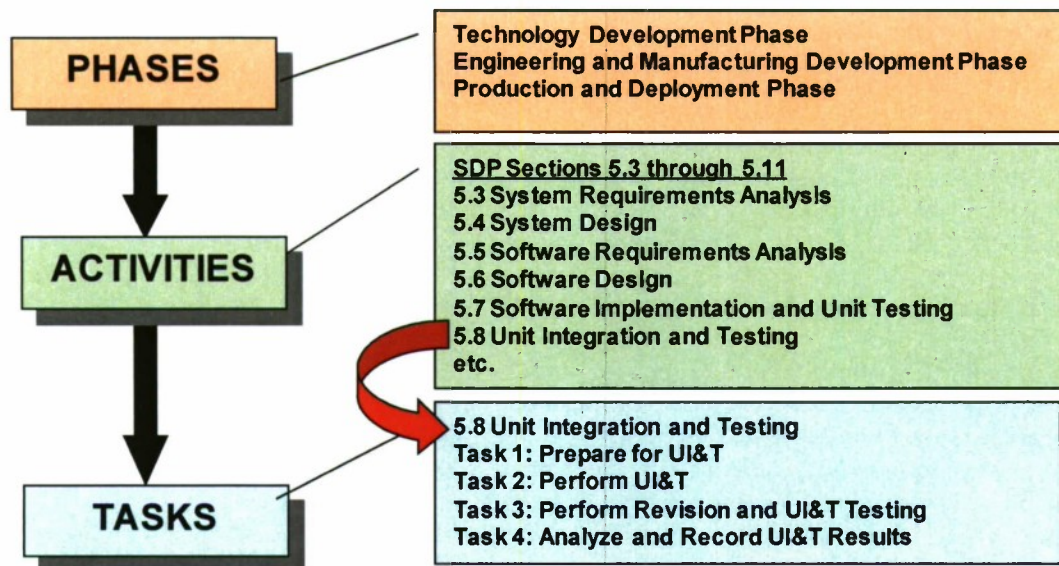


Figure 3.2. Software Process Levels Used In This Guidebook

### 3.3 Software Item Overview

Although not specifically required by TOR-3537B, an overview of the planned Software Items (SI) should be provided in this SDP subsection. An overview of the SIs that are to be developed can be best displayed in a table that defines the SIs in terms of the responsible Integrated Product Teams (IPTs), where in the system the SI is used, the developing organization, the programming languages used, and the software category for each SI. An example is shown in Table 3.3.

Table 3.3. XMPL Software Items and Team Responsibilities—Example

Software Item	IPT	System Element	Developer	Languages	SW Category
Spacecraft Controller Processor	Space	Spacecraft	Able Corp.	C	MC1, MC2
Payload Support Processor	Space	Spacecraft	Able Corp.	C	MC1
Vehicle Dynamic Simulator	Space	Spacecraft	Able Corp.	C	SS1
Data Management	Ground	Data Processor	Baker Corp.	C++, Java	MC1
Data Delivery	Ground	Data Processor	Baker Corp.	C++, Java	MC1
Infrastructure	Ground	Data Processor	Baker Corp.	C++, Java, IDL	MC1, SS1, C/R1
Calibration and Validation	Ground	Data Processor	Baker Corp.	C, C++, Java, Visual Basic	SS1, SS2, SS3
Satellite Operations	C3S	Mission Management Center	Charlie Corp.	C, C++, Java, FORTRAN	MC1, SS1, SS2, SS3, C/R1
Mission Management	C3S	Mission Management Center	Charlie Corp.	Java, C++	MC1, SS1, SS3
Ground Operations	C3S	Mission Management Center	Charlie Corp.	C, C++	MC1, SS1, SS2, SS3

Table 3.3 can become a very long table; in that case it should be included in an SDP Appendix and referenced in subsection 3.3. This table can be expanded with additional columns, such as percent new versus reuse code, and developer contact information.



### 3.4 Required Software Lifecycle Activities

Figure 3.4 is an example illustration of the required software activities during the software lifecycle development organized into four domains. Figure 3.4 also identifies the subsections within the SDP where each activity of the software development process is described.

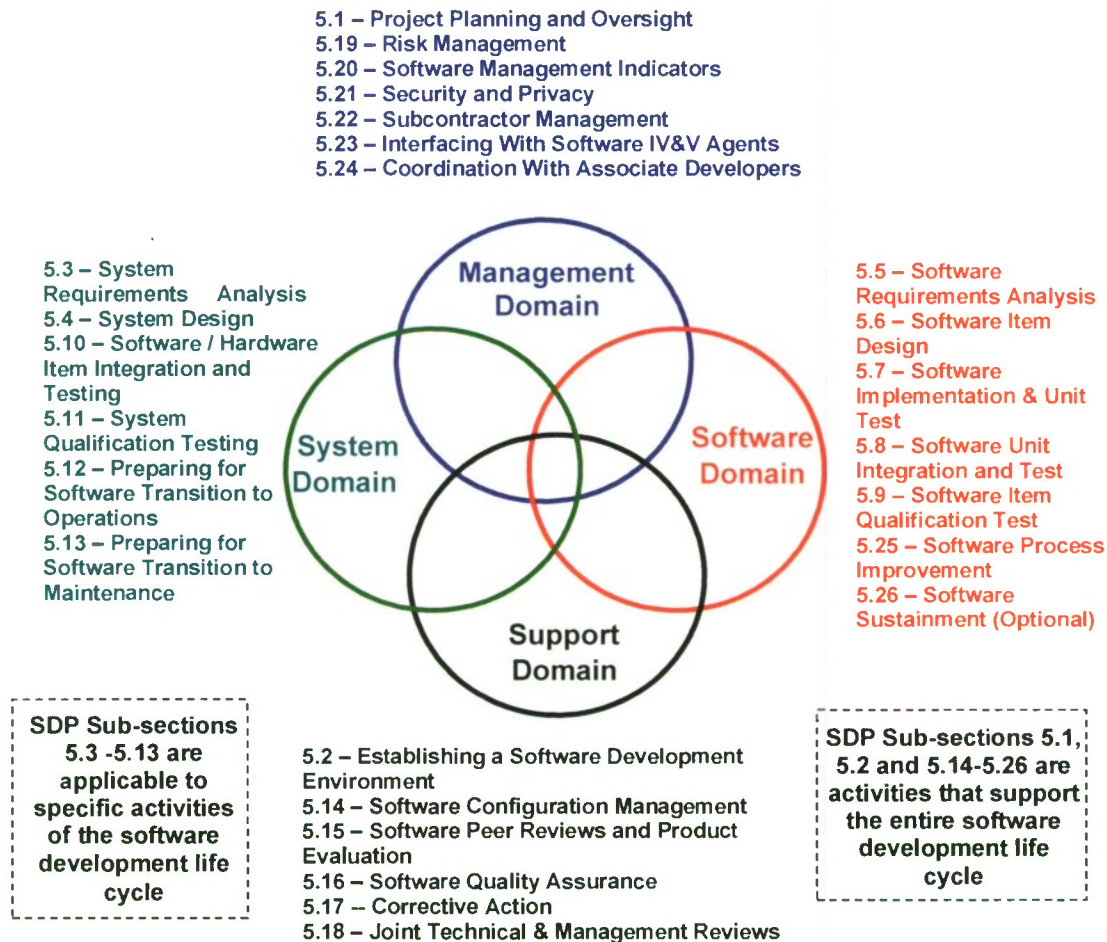


Figure 3.4. Software Lifecycle Development Domains—Example

### 3.5 Software Process Overview

In addition to overviews of the SIs and development activities in the previous subsections, it is recommended and extremely useful to include in subsection 3.5 an overview of the software development process that the program expects to follow and cover in more detail in SDP Section 4.

Figure 3.5-1 is one example of how to illustrate an overview of the software development process. It shows the principal software areas of responsibility as well as where software supports System Engineering for system-related activities. Figure 3.5-2 is a depiction of the specific software development activities and the sections of the SDP (subsections 5.3 through 5.13) where the activity is described. Although Figure 3.5-2 implies a sequential process, the actual process is dictated by the software development process model used (see SDP Appendix B) as well as an overlap of the activities consistent with the build plan.



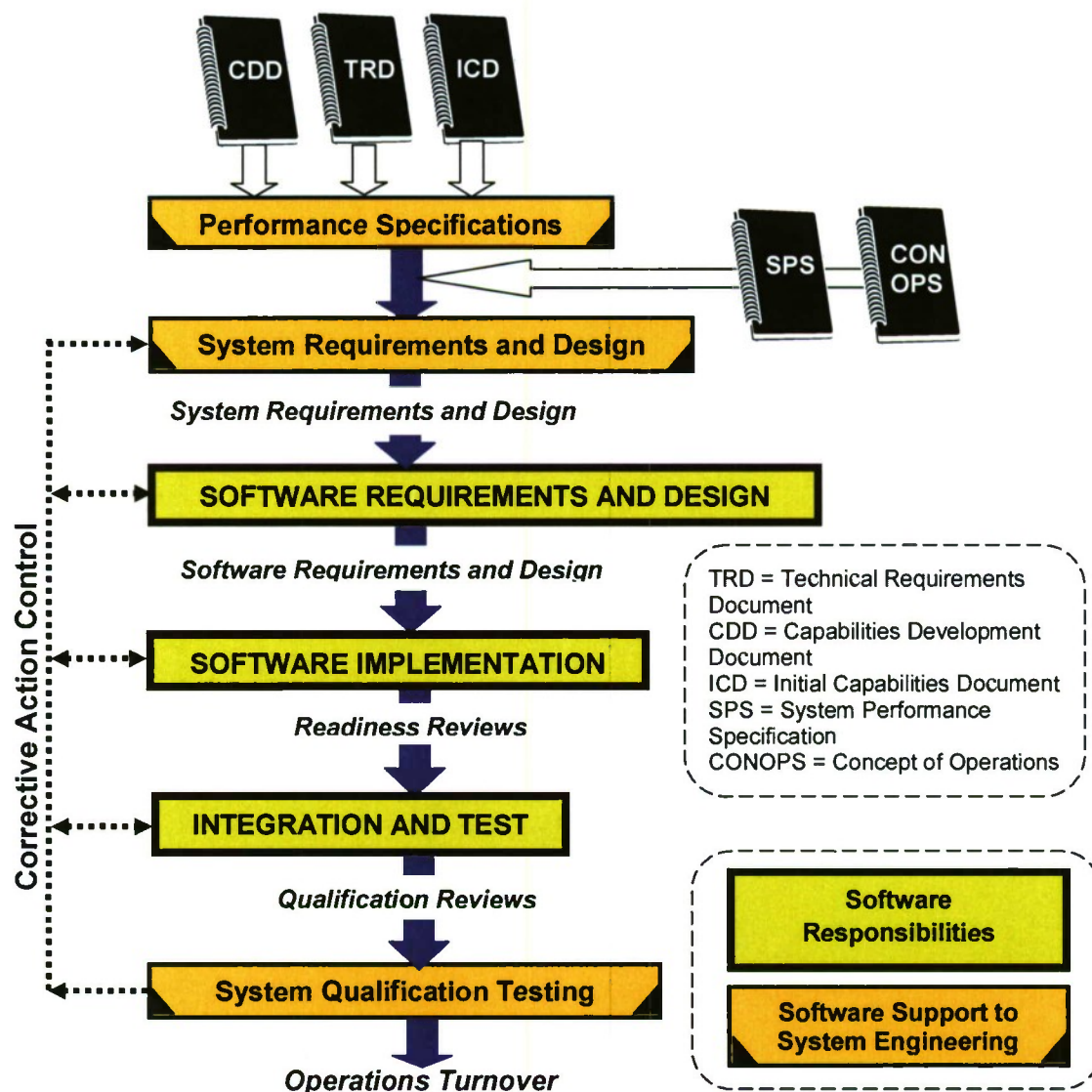


Figure 3.5-1. XMPL Software Development Process Overview—Example

### 3.6 Software Documentation Requirements and Constraints

During the software development process, various documents are required at different phases of the lifecycle. It is recommended and extremely useful to include an overview of the plans for production of software documentation in subsection 3.6. An example of a Software Documentation Production matrix is shown in Table 3.6.

The example document production matrix in Table 3.6 is an important guide as it summarizes the preparation of required work products (i.e., documentation) during the software development and test lifecycle covering SDP subsections 5.5 through 5.13. It identifies the normal preparation of draft (D), preliminary (P), and baselined (B) documents as well as when baselined documents are updated (U). Some documents that are prepared may not be required to be delivered. They may be prepared to be compliant with TOR-3537B but not contractually deliverable (such as unit test plans, descriptions, and reports). The contract **must** identify the required work products to be delivered.

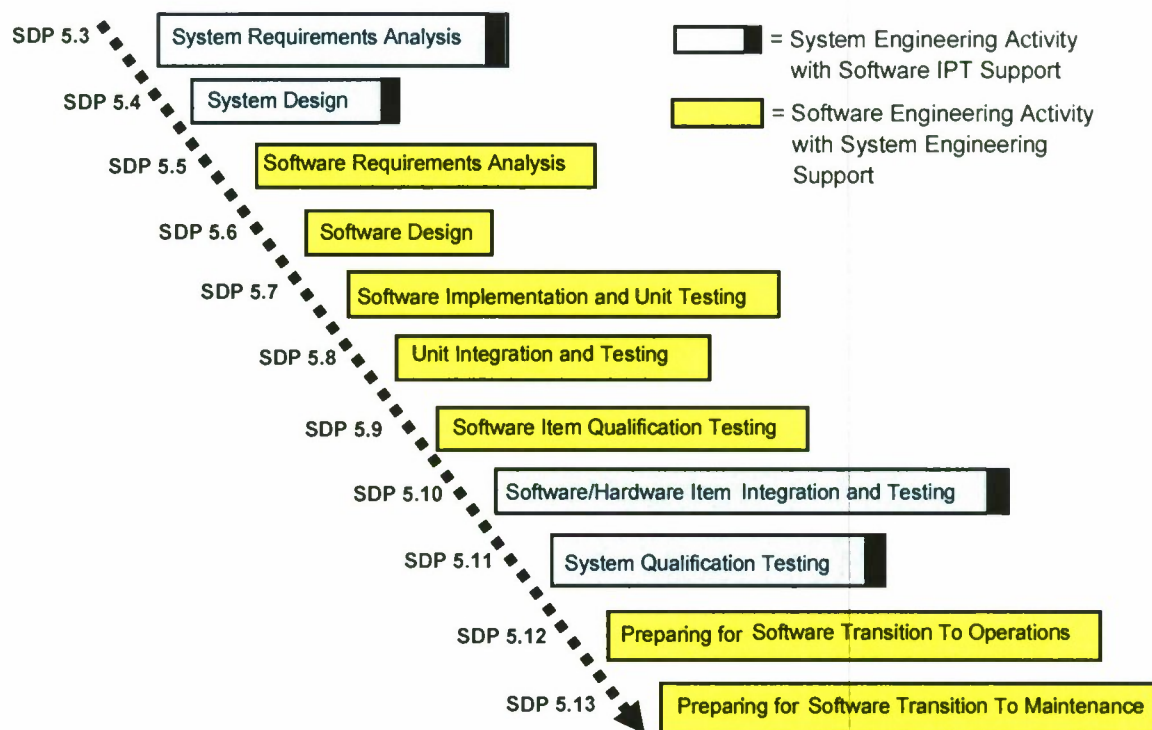


Figure 3.5-2. Principal Software Development Process Activities—Example

Documents, and other software products required at each activity of the lifecycle, are discussed in subsections 5.5 through 5.13, and the matrix **must** be consistent with the required work products tables appearing in each of those subsections. See Table 5.18.1-2 for a breakdown of software documentation mapped to formal reviews.

Non-document software work products, as defined in subparagraph 4.2.10.3, are not included in the documentation production matrix in Table 3.6.

In addition, Table 3.6 does not include software management and quality control plans such as the: Software Development Plan; Risk Management Plan; Data Management Plan; Subcontractor Management Plan; Software Safety Plan; Software Configuration Management Plan; Software Quality Assurance Plan; Software Process Improvement Plan; Software Peer Review Plan; Software COTS/Reuse Plan; Software Metrics Plan; Software Reviews Plan, etc. (see subparagraph 4.2.10.1).

It is also recommended to include in the SDP a master index of all software documentation. That index can be included as an SDP Appendix. For more information on software deliverable documentation see TOR-2006(8506)-5738, *Recommended Software-Related Contract Deliverables for National Security Space System Programs*, dated 14 February 2008.

Data Item Descriptions (DIDs) **must** be listed (as applicable) on the Contract Data Requirements List (CDRL) to ensure the software work products are delivered under the contract. TOR-3537B provides a list of the software DIDs. Each DID provides a full description of the contents of each deliverable software document. Annexes E through J in J-16 also provide a similar description of software document contents. Note that the Master Test Plan is not a software document.



Table 3.6. XMPL Software Documentation Production Matrix—Example

Software Development Activities (5)	Software Documentation										
	SRS	IFCD	SMBP	SDD SAD	IDD DBDD	STP	STD	STR	SVD SPS	SUM STrP	FSM CPM
Software Requirements Analysis	P (1)	B	D								
SI Architectural Design	B	U	B	P	D/P	B					
Software Item Detailed Design (2)	U		B	B	B	B				P	
Software Implementation and Unit Testing (2)	U				U	U					D
Unit Integration and Testing (2)						U	D/P		D		
SI Qualification Testing (3)						U	P/B	B	B	P	P
SI/Hi Integration and Testing (3)							U		B		
System Qualification Testing									U		
Preparing for SW Transition to Operations									U	B (4)	
Preparing for SW Transition to Maintenance									U		B

**MATURITY LEGEND:**

D = Draft In Process  
P = Preliminary Baseline Completed  
B = Baselined  
U = Updated Baseline (as needed)

**SOFTWARE DOCUMENTATION:**

SRS = Software Requirements Specification  
IFCD = Interface Control Document  
SMBP = Software Master Build Plan  
SAD = Software Architecture Description  
SDD = Software Design Description  
IDD = Interface Design Description

DBDD = Data Base Design Document

STP = Software Test Plan

STD = Software Test Description

STR = Software Test Report

SVD = Software Version Description

SPS = Software Product Specification

SUM = Software Users Manual

FSM = Firmware Support Manual

CPM = Computer Programming Manual

STrP = Software Transition Plan

SI = Software Item

- (1) In this example, the SRS contains the Interface Requirements Specification (IRS), Software Requirements Traceability Matrix (SRTM) and Requirements Test Verification Matrix (RTVM).
- (2) Iterative for each build.
- (3) This activity may be iterative, in reverse order, or concurrent.
- (4) Other optional user manuals include: Computer Operation Manual (COM); Software Center Operations Manual (SCOM); Software Input/Output Manual (SIOM).
- (5) The 'Development Activity' name is equivalent to the principal activity being performed at that time.



### 3.7 Requirements and Constraints on Development Strategy

#### 3.7.1 Development Strategy Factors

There can be many factors, and constraints, that impact the development strategy. For example, if the program involves a large number of geographically dispersed subcontractors from different companies, the overall approach to management and communication will have a significant impact on the development strategy and those issues need to be addressed. Another example involves programs that plan to utilize a significant amount of COTS/Reuse software. SDP paragraph 4.1.3 is devoted entirely to the management and implementation of COTS/Reuse software. However, its impact on the development strategy should be briefly addressed in this paragraph.

#### 3.7.2 Software Integration, Testing, and Verification Approach

Subsections 5.7 through 5.11 of the SDP describe the software Integration, Testing, and Verification (IT&V) activities. It is recommended, and would be extremely useful, to include in this paragraph of the SDP an overview of the software IT&V approach and process before describing the details in subsections 5.7 through 5.11. It **must** be stated that the software IT&V approach is consistent and compliant with the system-level integration and verification test plan (sometimes called the System Master Test Plan).

The rationale for software testing, described as an example in this Guidebook, is based on an incremental buildup of tested requirements with a simultaneous incremental verification buildup. The software IT&V process involves four generic testing stages as shown in Table 3.7.2.

Table 3.7.2. Software Integration, Testing, and Verification Stages—Example

Stage	Description
<b>Stage 1: Development Testing</b>	Stage 1 testing covers Software Unit (SU) testing and integration by the software developers, unit integration testing, and individual Software Item (SI) qualification testing. These stages of software I&T are covered in SDP subsections 5.7, 5.8, and 5.9.
<b>Stage 2: Element Testing</b>	Stage 2 testing includes: integration of multiple Software Items; integration of the Hardware Items (HI) with SIs; and the Element Acceptance Test (EAT) that may also be referred to as the "Factory Acceptance Test" (FAT). It normally takes place at the Segment Level depending on where the software entities are developed. The SI/HI integration is covered in subsection 5.10 of the SDP.
<b>Stage 3: Segment Testing</b>	Stage 3 of testing takes place in a location where elements are integrated and SI/HI elements are tested with other SI/HI elements. Generally, this stage can be viewed as the location where all of the elements of a segment come together. It includes the functions of Installation, Checkout and Test plus Interface Testing. This stage of software testing is normally concluded with a Segment Acceptance Test (SAT and is described in subsection 5.11 of the SDP.)
<b>Stage 4: System Testing</b>	Stage 4 of testing is focused on the process of integrating all of the segments (and sites) into the full system or portions of the full system being tested. This stage of testing is normally concluded with a System Qualification Test (SQT) and is also described in subsection 5.11 of the SDP. Software has a <u>support role</u> in segment and system testing as those activities are typically the responsibility of (SEIT).

### 3.7.3 Software Integration, Testing, and Verification Objectives

The objectives of each of the above four stages of the software IT&V process are summarized in example Table 3.7.3. That table identifies the subsection of the SDP containing details of the testing process at each stage and highlights key functions at each step of the IT&V buildup.

Table 3.7.3. Software Integration, Testing and Verification Objectives—Example

SDP Subsection and Title	Integration, Test, and Verification Objectives
<b>Stage 1a:</b> <b>5.7 Software Implementation and Unit Test</b>	<ul style="list-style-type: none"> <li>Convert Software Unit (SU) design into computer source code, compile, and debug</li> <li>Test/Inspect to ensure source code is compliant with expected results</li> <li>Verify that the source code meets the design</li> </ul>
<b>Stage 1b:</b> <b>5.8 Unit Integration and Testing</b>	<ul style="list-style-type: none"> <li>Integrate SUs that have successfully passed Code and Unit Test (CUT) and build them up to higher level SUs and to a SI</li> <li>Assure SUs are successfully integrated for the current build</li> <li>Perform design inspection through functional testing for current build</li> <li>Perform initial SI to SI interface testing, with stubs, drivers, or current SIs</li> </ul>
<b>Stage 1c:</b> <b>5.9 Software Item Qualification Testing</b>	<ul style="list-style-type: none"> <li>Demonstrate that the SI(s) satisfies the software and interface requirements</li> </ul>
<b>Stage 2:</b> <b>5.10 Software/Hardware Item Integration and Testing</b>	<ul style="list-style-type: none"> <li>SI to SI Integration and Testing integrates individual SIs of an element or segment to produce a complete software segment build</li> <li>SI to HI Integration and Testing integrates software with hardware</li> <li>Element Acceptance Test (EAT), verifies that: (a) software and hardware functional requirements defined in the element specifications, have been satisfied; and (b) functional and physical interface requirements have been satisfied for the current build</li> </ul>
<b>Stage 3:</b> <b>5.11 Segment Qualification Testing</b>	<ul style="list-style-type: none"> <li>Segment Acceptance Test (SAT) verifies that the segment hardware and software functional and interface requirements have been satisfied</li> </ul>
<b>Stage 4:</b> <b>5.11 System Qualification Testing</b>	<ul style="list-style-type: none"> <li>System Qualification Test verifies that the system performance specifications and all interface requirements (functional, physical, and external) have been satisfied for the entire system or that portion of the full system being tested</li> </ul>

### 3.7.4 Software Integration, Testing, and Verification Process

It is recommended, and it would be extremely useful, to include in paragraph 3.7.4 an overview of the software IT&V process. An example of the software integration and testing process is graphically depicted in Figure 3.7.4. The figure is, and **must** be, consistent with paragraph 3.7.2 and Table 3.7.3.

### 3.8 Requirements and Constraints on Schedule and Resources

SDP Section 6 covers the program schedule and activity network, and SDP subsection 7.2 is focused on project resources. However, TOR-3537B mandates a brief discussion of key requirements and constraints, for both schedule and resources, in SDP Section 3. References must be made to the program's Integrated Master Schedule (IMS) and all software schedules must be consistent with the IMS.

Reference should also be made to the Contract Data Requirements List (CDRL) Form 1423 that identifies documentation and product content, format, and delivery schedule. This subsection should also be used to identify who is responsible for software compliance with the IMS delivery schedule.

### 3.9 Other Requirements and Constraints

Subsection 3.9 can be anything it needs to be to define other actual or potential requirements and constraints. Frequently, this subsection is organized in two paragraphs: Contractual Constraints and Non-Contractual Constraints. The following example text may be used as a guide:

**Example Text:****3.9.1 Contractual Constraints**

During System Acquisition phases, the software activities are constrained by the XMPL IMP, IMS and System Specification. Technical Operating Report, TOR-2004(3909)-3537B, "*Software Development Standard for Space Systems*" will be used as a compliance document for software processes, and organization of this SDP.

**3.9.2 Non-Contractual Constraints****3.9.2.1 Company Policies and Practices**

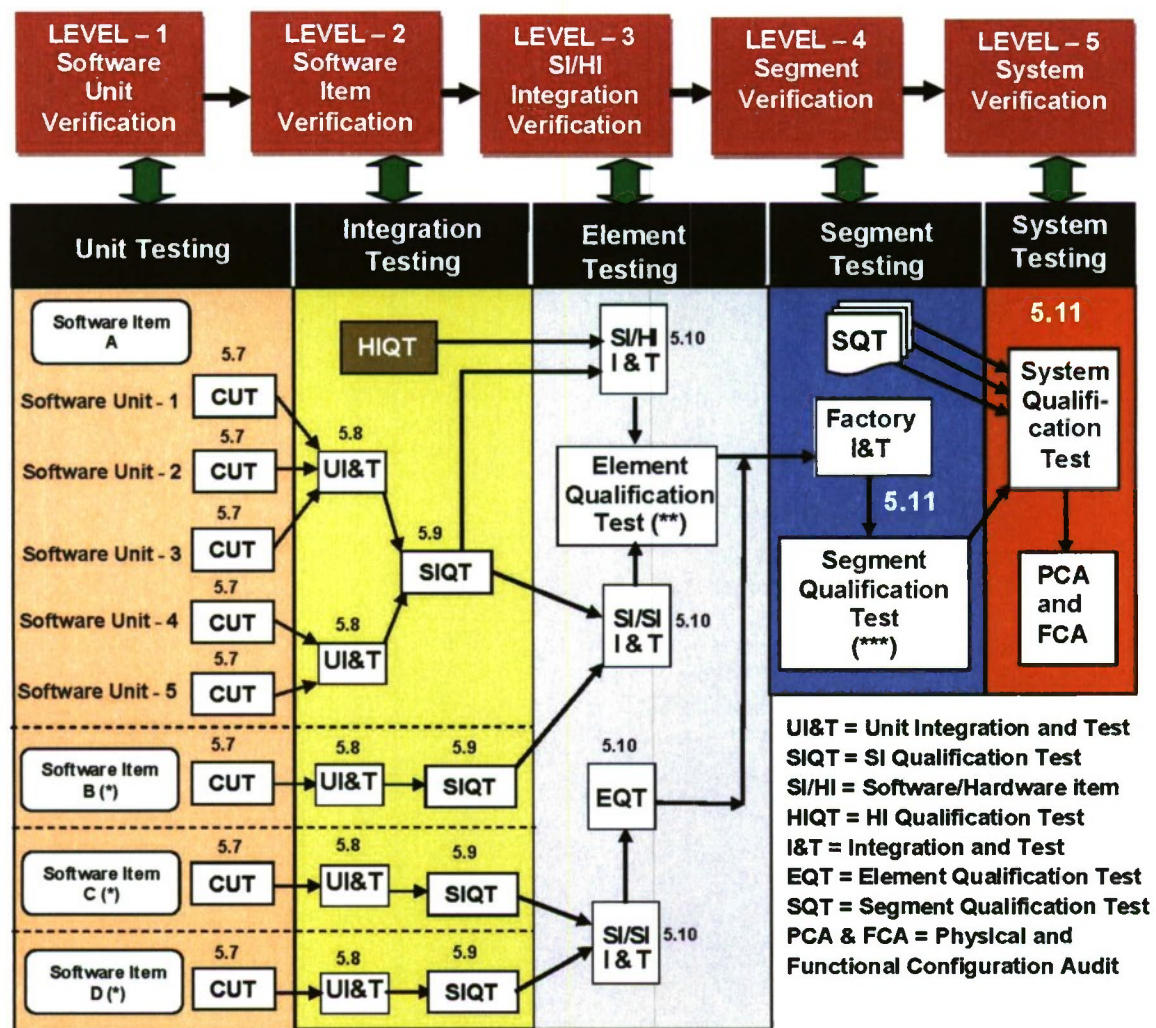
Additional constraints on the XMPL software development process are levied by virtue of compliance of this SDP with the <corporate> standard software process and the related <corporate> Standard Software Process Manual. The corporate software policies are based on commercial standards such as ISO 9001 as well as the SEI's CMMI<sup>®</sup>.

**3.9.2.2 XMPL Program Policies and Practices**

Software development activities are also constrained by key program plans and approved program procedures. The key XMPL program plans include the:

- Software Development Plan (SDP)
- Contract Implementation Plan (CIP)
- Configuration Management Plan (CMP)
- Risk Management Plan (RMP)
- System Engineering Management Plan (SEMP)





(\*) Breakdown of Software Units for Software Items B, C & D are not shown

(\*\*) Some contractors call this Element Acceptance Test (EAT) or Factory Acceptance Test (FAT)

(\*\*\*) Some contractors call this Segment Acceptance Test (SAT)

Figure 3.7.4. Software Testing and Integration Process—Example

## 4. General Requirements

To be compliant with TOR-3537B, Section 4 of the SDP **must** include two subsections:

- Subsection 4.1 **must** contain an overview of the software development process to be used including the process for each software class, the lifecycle software model(s) to be used, and the plans for software builds including the software development activities to be performed for each build. Although there is no specific organization for subsection 4.1, paragraphs 4.1.1 through 4.1.3 as described below are recommended as the minimum content for this subsection.
- Subsection 4.2 **must** cover general plans for software development in eight paragraphs as required by TOR-3537B. Two additional optional additions to subsection 4.2 are recommended:
  - 4.2.1 Software Development Methods
  - 4.2.2 Standards for Software Products
  - 4.2.3 Traceability
  - 4.2.4 Reusable Software Products
  - 4.2.5 Assurance of Critical Requirements
  - 4.2.6 Computer Hardware Resource Utilization
  - 4.2.7 Recording Rationale
  - 4.2.8 Access For Acquirer Review
  - 4.2.9 Software Data Management (Recommended Optional Addition)
  - 4.2.10 Software Work Products (Recommended Optional Addition)

### 4.1 Software Development Process

Software development lifecycle models should be used to describe, organize, and monitor the software development activities. A detailed discussion of the various models (e.g., Waterfall, Incremental, Evolutionary, Spiral, and Unified) is beyond the scope of this Guidebook; however, Table 4.1 is an overview of the most commonly used software development process models. Each program **must** select the strategy appropriate to the software being developed and that process **must** be defined in the SDP. More than one software development lifecycle model may be needed for different types of software.

The example software development process described in this Guidebook is an incremental (or multi-build) development approach using the development processes defined in TOR-3537B as a guideline. A graphical overview is recommended as the ideal approach to depict the software development process. Separate process charts should be created for Mission Critical and Support Software classes and, if necessary, separate figures for the categories within the classes (as described in paragraph 1.2.3).

#### 4.1.1 Mission Critical Software Development Process

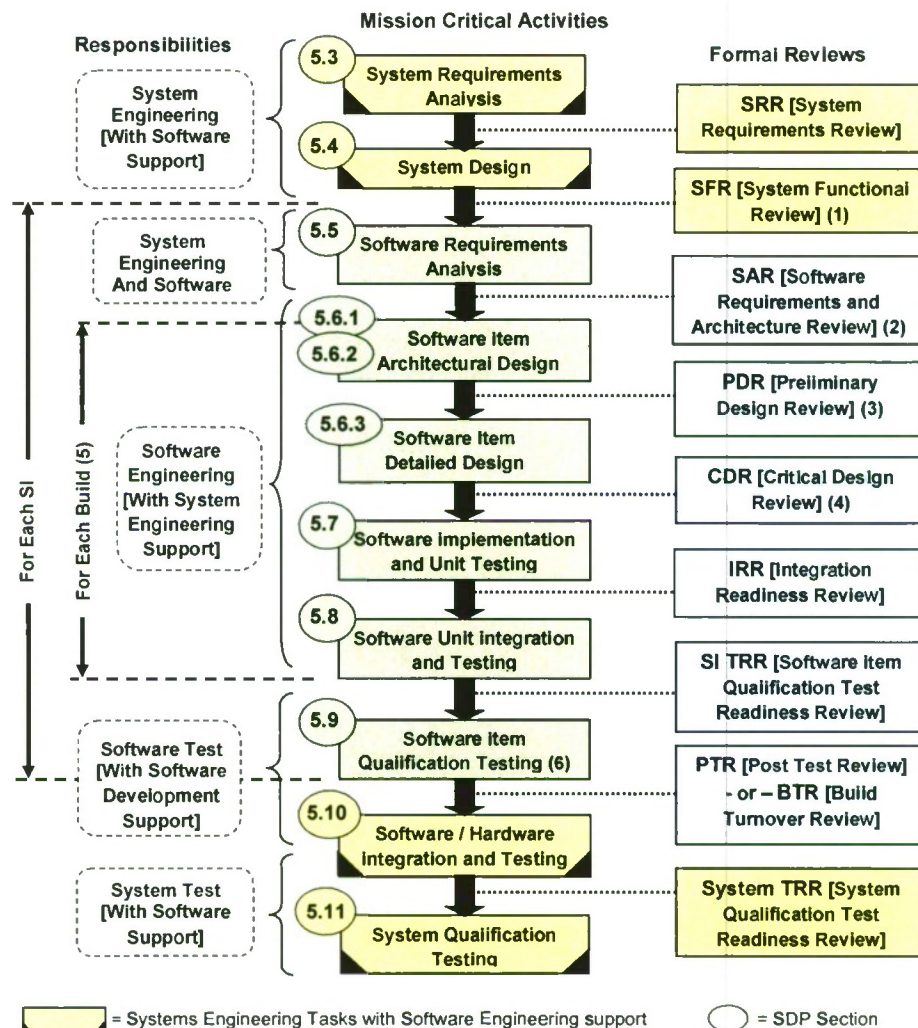
Figure 4.1.1 is an example graphical overview of a software development process for Mission Critical (MC) software as described in subparagraph 1.2.3.1. The MC software process is the most comprehensive process since it provides critical application software functionality. It **must** support development of as many SIs and builds as needed to meet program milestones.



Table 4.1. Overview of Software Development Process Models—Example

Process Model	Description
<b>Rapid Prototyping</b>	This development approach involves building an early experimental system, or portions of the system, to better understand the requirements and interfaces, to test throughput speeds, develop environment testing, etc. Since the product produced is built fast, without sufficient documentation, and not designed to be maintainable, it cannot be used as the final product.
<b>Waterfall</b>	This is a sequential software development model that requires each activity to be completed before the next activity begins, although some overlap is allowed. The requirements and design activities are defined up front. The entire functional software product is not available until the last testing activities are completed.
<b>Incremental</b>	This model requires that all of the requirements must be defined up front; the software product is then developed in a series of builds, or blocks, with increasing functionality. A portion of the software product is built and tested—one small increment at a time. This is a “build-a-little, test-a-little” approach that can provide an early operational capability for a portion of the entire system.
<b>Evolutionary</b>	With this model the software product is developed in a series of builds, or blocks, with increasing functionality. However, the requirements are defined for each evolutionary build as that build is developed. This is also a “build-a-little, test-a-little” development process model that can provide an early operational capability for a portion of the entire system, and it is highly amenable to evolving requirements.
<b>Spiral</b>	This is a risk-driven software development process model that has two main features: (1) A cyclic approach that grows a system's functionality and implementation incrementally while focusing on decreasing its degree of risk; and (2) A set of anchor point milestones for insuring stakeholder commitment to acceptable system solutions. Implementations using this model are often done in conjunction with either the incremental or the evolutionary model.
<b>Unified Process</b>	A variation of the Spiral Model is the Unified Process exemplified by the IBM Rational Unified Process® (RUP®). RUP is an iterative software development framework. However, it is not a single prescriptive process but an adaptable process framework intended to be tailored by selecting elements of the process applicable to each user. It has an underlying object-oriented model using the Unified Modeling Language (UML).





- (1) Also called the System Design Review (SDR).  
 (2) SAR and PDR may be combined for object-oriented development because requirements definition and architectural design are usually iterative. The SAR was formerly called the Software Specification Review (SSR).  
 (3) An optional SBRAR (Software Build Requirements & Architecture Review) may also be held in addition to the PDR.  
 (4) An optional SBDR (Software Build Design Review) may also be held in addition to the CDR.  
 (5) This range is for the Incremental Development Model; the Evolutionary Model would extend to activity 5.5.  
 (6) Software Qualification Testing may be done within each build.

Figure 4.1.1. Mission Critical Software Development Process—Example

The following example text may be used for paragraph 4.1.1:

**Example Text:**

As shown in Figure 4.1.1, the MC software development process begins with requirements definition for each XMPL SI using system-level documents such as the Technical Requirements Document (TRD) and segment-to-segment Interface Specifications. Requirements from these specifications are allocated to software and hardware, and the allocated software requirements are decomposed, elaborated, and documented in the Software Requirements Specification (SRS) and Interface Requirements Specification (IRS). For this iterative lifecycle model, detailed design, code, integration, and test activities are performed for each SI within a build. Once the SIs are integrated and tested for a build, the build is delivered, with the Software Version Description (SVD), to the cognizant software development library for Configuration Management control.

### 4.1.2 Support Software Development Process

Figure 4.1.2 is an example graphical overview of a software development process for Support Software (SS) as described in subparagraph 1.2.3.2. Although SS operates only in non-operational environments, the SS-1 category normally requires the same level of documentation as MC software. However, reviews for Support Software may not be as formal or as frequent. The principal differences between the examples for MC (Figure 4.1.1) and SS-1 (Figure 4.1.2) processes are:

- There is no formal SSR, PDR, CDR, IRR, PTR, and BTR as shown for MC software in Figure 4.1.2.
- The formal reviews are replaced by Technical Interchange Meetings (TIMs)
- Architecture and detailed design phases are merged and followed by a TIM

The SS-2 development process, as described in subparagraph 1.2.3.2, should be expected to be similar to the SS-1 process, but less stringent, usually having principal differences such as the following:

- SS-2 requirements information is normally maintained in a requirements database and referenced in SDFs, just as it is for SS-1. However, a formal SRS document may not be required.
- Design material is maintained in the SDFs, just as it is for SS-1. However, a formal SDD and IDD is not usually required.
- Informal SI and software build test descriptions and test results are maintained in SDFs. However, formal STP, STD, and STR documents are not usually required.
- TIMs performed for SS-1 may be replaced by Peer Reviews for the inspections and verifications of work products developed for SS-2.
- Applicable software metrics data should be collected for SS-2 software. However, the metrics data set and the reporting frequencies may be reduced.

### 4.1.3 Iterative Process

The vertical arrows between the phase boxes in Figures 4.1.1 and 4.1.2 are misleading as they imply a straight through process. In reality, it is an iterative process with corrective action control loops as depicted graphically in the process overview Figure 3.5-1. The selected software lifecycle model(s) for the planned software task should be described in subsection 4.1 with a description of the consistency between the software development model(s) and the system lifecycle model(s).

If there are software builds, there is a requirement in TOR-3537B, as part of the description of the process to be used, for subsection 4.1 to identify the planned software builds, their objectives and the software development activities to be performed in each build. Those issues are covered later in subparagraph 5.1.1.3 Software Build Planning.

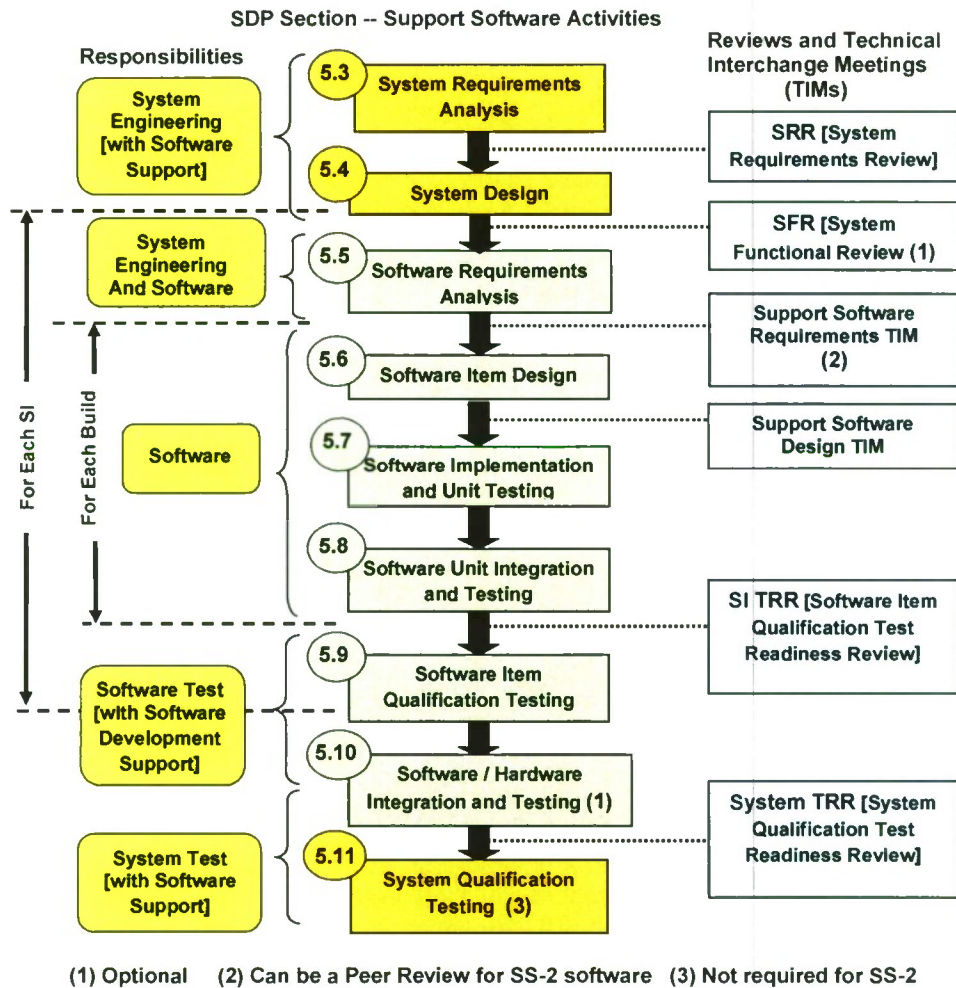


Figure 4.1.2. Support Software Development Process—Example

## 4.2 General Requirements for Software Development

The following paragraphs and subparagraphs describe general requirements for software development. Details of the software development process activities **must** be provided in SDP Section 5, and in related tables and addenda. Software test and integration details can also be provided in an addendum.

### 4.2.1 Software Development Methods

The software development method(s) to be employed **must** be described, or referenced, in SDP subparagraph 4.2.1. As noted in TOR-3537B, “automated tools and procedures to be used in support of these methods” need to be described. However, references should be made to SDP subsection 5.2 where automated tools are discussed in depth as part of the Software Development Environment (SDE). The following example text may be used as a guide:



**Example Text:**

XMPL software development will follow an object-oriented (OO) methodology. The OO methodology includes Object Oriented Analysis (OOA) and Object Oriented Design (OOD) utilizing the Unified Modeling Language (UML) notation. Segments must declare in their SDP Annex which methodology will be used if not using OO. The UML notation standards and programming standards for C++ and Java are defined in the Appendices to the program-level SDP. Structured analysis and design will be used for applications such as coding scientific/mathematical algorithms or emulation models and applications involving extremely data-intensive or high performance computing.

## 4.2.2 Standards for Software Products

The following bullets are references to software standards and practices that **must** be addressed in the SDP:

- COTS/Reuse software **must** adhere to requirements definition and testing processes, standards, and practices as specified in SDP paragraph 4.2.4 or in the program's Software COTS/Reuse Plan.
- The programming language standards to be used **must** be defined in an Appendix or Addendum to the SDP or preferably in a Software Standards and Practices Manual as discussed in subparagraph 4.2.2.1. That manual can also contain standards for architecture/design, software requirements, and software test documentation.
- Operational details of the program's defined software process should be elaborated through detailed Work Instructions and/or Procedures (see subparagraph 4.2.10.2). Relevant Work Instructions and Procedures should be listed in an Appendix or Addendum to the SDP if the list is long.
- Software development for the program **must** be guided by the applicable standards listed in Section 2 of the SDP.

**Hierarchical Software Product Levels.** The hierarchy of software related specifications **must** be produced in accordance with the program's Specification Tree. Typical hierarchical software product levels and terminology are depicted in Figure 4.2.2. References to a "Software Unit (SU)" in this SDP Guidebook can be interpreted as a single SU or as a group of integrated SUs as applicable.

### 4.2.2.1 Software Standards and Practices Manuals

Standards for software requirements, architecture, design, code, and test **must** be documented. The recommended location for these standards is in Software Standards and Practices Manuals that can be addenda to the SDP. These standards ensure that developers produce consistent software development products. Coding standards, for example, can include standards for formatting, comments, naming conventions, and restrictions on programming language constructs and features. Standards also help ensure the similarity of the structure of all code/design units so that lines of code counts and software measurements can be applied consistently.

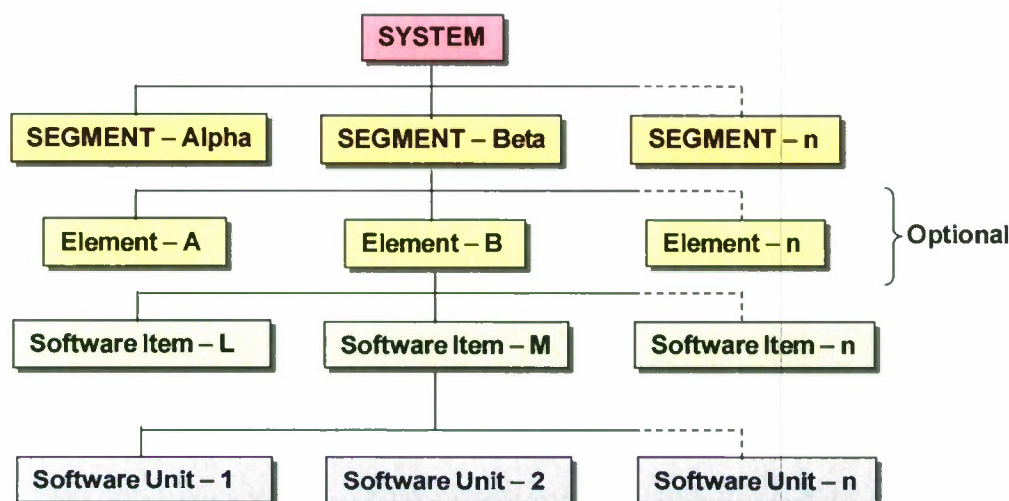


Figure 4.2.2. Hierarchical Software Product Levels—Example

The SDP should also document the process for waivers or deviations to these standards. Changes **must** be justified, documented, and submitted by the cognizant Software Item Lead, approved by the IPT Lead and SQA, submitted to the Software Engineering Process Group (SEPG) for concurrence, and made part of the appropriate Software Development Files (SDF) or Software Engineering Notebook (SEN). The SEPG should review software standards and tools usage and provide the means for sharing knowledge and lessons learned across the program and with the SEPGs at the segment or system level.

### 4.2.3 Traceability

An automated traceability and requirements management database **must** be used by every large software intensive program. Examples of such tools are: Dynamic Object-Oriented Requirements System (DOORS), System Level Automation Tool for Engineers (SLATE), Requirements and Traceability Management (RTM) and Requisite Pro. In this Guidebook, the requirements management and traceability tool will be called the "Requirements Database." Table 4.2.3 is an example of traceability products that should be produced for each software category.

TOR-3537B requires the SDP to describe the approach to be followed for establishing and maintaining bi-directional traceability between:

- Levels of requirements
- Requirements and design
- Design and the software that implements it
- Requirements and qualification test information
- Required and measured computer hardware resource utilization

Table 4.2.3. Traceability Requirements by SI Category—Example

Software Requirements Traced to:	MC-1 and MC-2	SS-1	SS-2 and SS-3	CR-1 and CR-2
Parent Requirements	Required in tool	Required	Required	Required
Software Builds	Required	Required	Required	Required
Use Cases	Required	Required	Not Required	Not Required
Software Units	Required	Required	Required	Not Required
Software Test Cases	Required	Required	Required	Required
Software Test Procedures	Required	Required	Not Required	Required



#### 4.2.4 Reusable Software Products

The term “reusable software product” is normally defined as any existing software product (i.e., specifications, designs, test documentation, executable code, and source code) that can be effectively used to develop the software system. COTS/Reuse software has become much more important and widely used over the past decade. **Programs that plan to use a significant amount of COTS/Reuse software must address COTS/Reuse in considerable detail in their SDP.**

Reusable software products may include software that is not modified, migrated software that requires changes, and newly developed software usable in other application areas of the program. Software development teams should consider the use of reusable software products wherever possible. Reusable software products can include Commercial Off-The-Shelf (COTS) and Government Off-The-Shelf (GOTS) software products as well as reuse libraries.

Two options are suggested for addressing the COTS/Reuse issues in the SDP: (1) cover all of the topics in SDP paragraph 4.2.4; or (2) include an informative overview in SDP paragraph 4.2.4 and refer to a Software COTS/Reuse Plan for the details. This Guidebook favors the second option.

Two subparagraphs, 4.2.4.1 and 4.2.4.2, are required by TOR-3537B. They should provide an overview and point to the Software COTS/Reuse Plan for the details. That plan should be an addendum to the SDP and should include a discussion of the following reuse topics:

- Establishing and managing the Software COTS/Reuse Plan
- Heritage reuse base programs
- Controlling, testing, and upgrading COTS/Reuse baselines
- Developing and integrating reusable software products
- Approach to managing COTS/Reuse software implementation
- COTS/Reuse software selection criteria and responsibilities

##### 4.2.4.1 Incorporating Reusable Software Products

The approach to be followed for identifying, evaluating, and incorporating reusable software products **must** be described in this subparagraph. It **must** include the scope of the search for such products, the criteria to be used for their evaluation, and address all of the related contractual clauses. If reusable software products have been selected, or identified at the time the SDP is prepared or updated, they **must** be identified and described including their known benefits, risks, constraints, and restrictions. The SDP should cover the entire COTS/Reuse lifecycle, including identification, investigation, evaluation, selection, implementation and maintenance as depicted in example Figure 4.2.4.1.



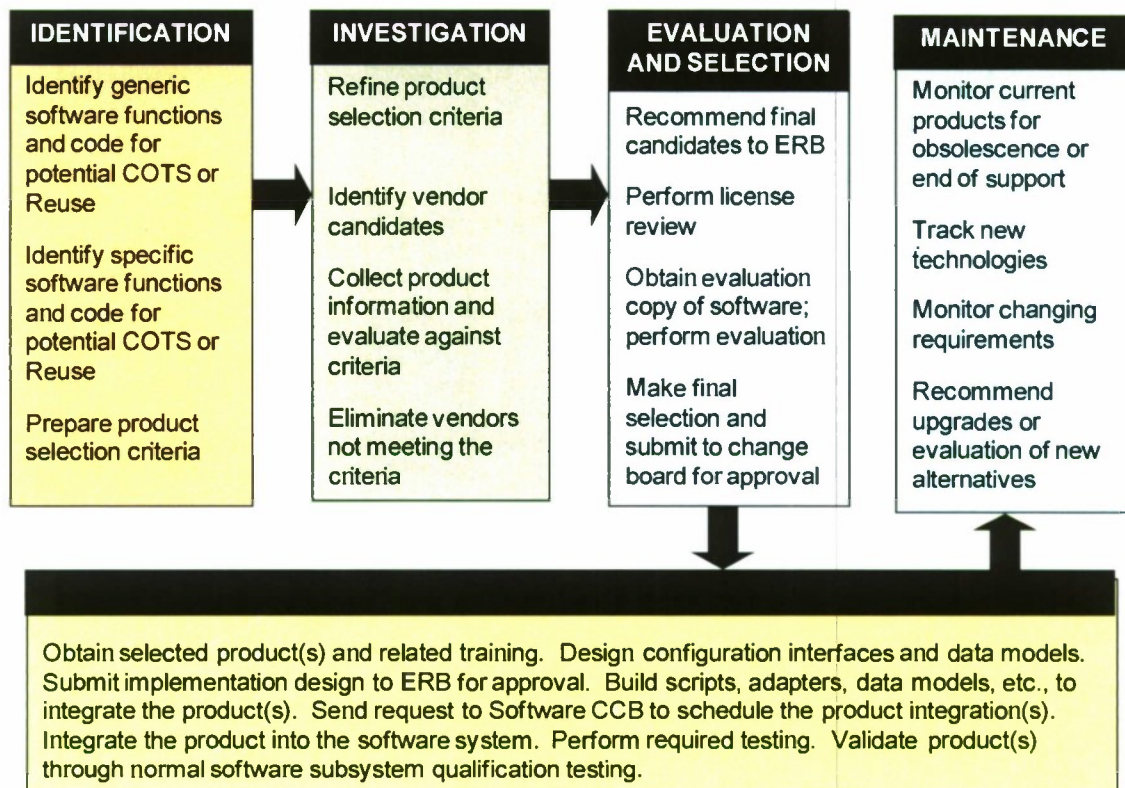


Figure 4.2.4.1. COTS/Reuse Management Process—Example

**Reusable Software Criteria.** Reusable software products **must** meet the specified technical and contractual requirements and be cost-effective over the life of the system. The following factors should be considered in an evaluation of candidate reusable software products:

- Technical capabilities or applicable functionality
- Safety, security, and privacy requirements
- Demonstrated reliability and product maturity
- Testability and availability of test cases and data
- Short- and long-term cost impacts of using the software product
- Technical, cost, and schedule risks and tradeoffs in using the software product
- Data rights transferable to the software product
- Interoperability with target software environment
- Availability and quality of documentation and source files
- The need for required changes and the feasibility of making those changes
- Supplier maintainability and warranty
- Restrictions on copying/distributing the software or documentation

Note that a bad evaluation report on any single factor can be a sufficient condition to reject a reuse candidate. Appendix B of TOR-3537B contains criteria for evaluating reusable software products.

**Approach to Using COTS Software.** Using COTS software allows developers to be selective in what functions and capabilities can be acquired without having to pay the price for custom development. The use of COTS software can also have a major impact on the reduction of schedule risk and cost risk. However, the process of including COTS components is often difficult and care **must** be taken to avoid a number of potential risks.

If a COTS product requires modification of the code, it is no longer considered a COTS product. It becomes the responsibility of the contractor unless the vendor is hired to make the modifications and that can be an expensive and risky approach. Generally, if any COTS or reused product requires more than 30 percent recoding, it is usually more cost-effective to build it from scratch. Industry estimates for this threshold ranges from 15 to 35 percent.

The principal risk is the loss of control over the formalized development process when COTS products are acquired. Software vendors have their own agendas that are different from those who adopt their tools. Therefore, a tradeoff **must** be made to enjoy the benefits from using COTS software. To be successful in using COTS software the following major factors **must** be considered.

**COTS Software Functionality.** A selected COTS product may not have the exact functionality required to be responsive to specific allocated requirements. The COTS product may have more capability than is needed, or may not provide all the required functionality, thus necessitating integration with other components or making potentially sophisticated modifications. Key COTS-related questions include:

- How mature is the COTS product and how easy is the COTS product to use?
- Are the COTS product capabilities and operation fully understood?
- How are allocated requirements not satisfied by the COTS product handled?
- How are unneeded capabilities of the COTS product handled?
- How have known problems in the COTS product been rectified?

**COTS Software Integration.** Tradeoffs may be necessary because the constraints and requirements imposed by the selected COTS products typically results in less flexibility available to the software architect. The method of integrating selected COTS components may impose additional constraints on the architecture, and planners **must** account for the additional effort required to understand the behavior of the COTS products. Key COTS-related questions may include:

- Was the software architecture designed first and the COTS products selected to fit it?
- Is the development team trained and qualified to integrate the COTS product?
- Does the COTS product have an Application Programming Interface (API), and does the development team understand the API's capabilities and complexities?
- Have the impacts of the COTS product on system resources been analyzed?
- Has the size of the integration effort for the COTS product been estimated, and what is the level of confidence for the estimate?

**Management of COTS Implementation.** The implementation of COTS products introduces new issues that do not exist when an entire system is developed in-house. For example, licensing will have to be considered as well as other vendor relationships. Also, the cost of adopting and adapting the components **must** be considered as well. When all factors are considered, it may be more cost effective to build than to buy. Key COTS-related questions include:

- Was the COTS product selected using a defined selection and evaluation process?
- Have all the integration and related costs been properly estimated?
- How long has the vendor been in business and what is its financial stability?
- What relationship does the <corporate> team have with the vendor?
- Have the vendor's technical support capabilities been fully evaluated?
- Is the vendor willing to modify the product to meet the requirements? [Note: Requesting the vendor to modify their COTS product to meet the needs of the program is generally considered a high risk and is not recommended].



- Have mutual non-disclosure agreements and data rights been negotiated?
- Have cost-effective licensing agreements been worked out with the vendor?
- Has configuration management of the COTS product been properly planned for?
- Has integration testing of the COTS product been thoroughly planned?
- Have the risks related to using the COTS product been identified and managed?

**Reusable Software Responsibilities.** The Software IPT should be responsible for identification and evaluation of reusable software products for the SIs and SUs of the system. Beginning in the software requirements definition activity, and continuing through the testing activity, the Software IPT should identify appropriate candidate reuse products for each software activity.

Depending on the specific functionality being considered for reuse, the Software IPT may need to perform trade studies or perform some modeling or analysis with the candidate products to determine sufficient information to make an evaluation. If any technical or non-technical issue is not fully resolved prior to the point that the product is selected for use, the Software IPT **must** define the issue as a risk and resolve it before a final selection is made.

#### 4.2.4.2 Developing Reusable Software Products

In addition to reusing existing software products, there may be opportunities for new software products developed that can be used elsewhere. The Software IPT should carefully review the SIs under development for opportunities where software products can be used elsewhere to improve efficiency of the software development effort.

The use of object-oriented design naturally produces cohesive objects that encapsulate functionality and data, have well defined interfaces, and are therefore suitable for reuse in many instances. In addition, class hierarchies and design patterns capture commonality and provide for abstractions that can lead to reuse. Specific activities in software analysis and design processes identify opportunities for not only design and code reuse, but also use case and scenario reuse for requirements traceability and testing. These opportunities for reuse should be recorded in the design documentation.

The task of identifying, evaluating, and reporting opportunities for developing reusable software products is often tailored out in the typical environment addressed by this Guidebook.

#### 4.2.5 Assurance of Critical Requirements

Critical strategies **must** be identified in the SDP to ensure that software groups provide additional oversight and focus on incorporating critical requirements into the SIs. There are always some key software requirements that are critical cornerstones for safety, security, privacy protection, reliability, maintainability, availability, performance, etc. Strategies **must** be developed and employed to ensure that these critical requirements are satisfied

The strategies **must** be documented in the SDP, including both test and analyses, to ensure that the requirements, design, implementation and operating procedures, for the identified computer hardware and/or software, minimize or eliminate the potential for violating the established mitigation strategies. The SDP should also indicate how evidence is to be collected to prove that the assurance strategies have been successful.

The following five subparagraphs may be used as a starting point for developing specific details on the approach to be used by the program for handling these critical requirements.



#### 4.2.5.1 Software Safety

Safety requirements involve SIs or SUs whose failure may result in a system condition that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment. Each software-related safety-critical requirement identified **must be documented in the Safety Requirements section of the SRS and identified by a unique product identifier**. If aviation safety standards are specified in the contract as compliance documents, this subparagraph **must** describe the approach for complying with those standards.

The activities required for ensuring that safety-critical software requirements are met for the program **must be shared between the System Safety group, at the program level, and the segment software team**. Each IPT should assign responsibilities for safety issues and for coordination with System Safety. The software team is responsible for developing system software that is safe to operate and compliant with all appropriate safety standards and requirements.

The general approach to managing software safety-critical development activities for the program should be to integrate safety management into the software lifecycle activities. System Safety should play an integrated role in the software development process and the CCBs. This provides System Safety with visibility into the software development activities that are critical to program safety issues, and provides the IPTs with the input required to ensure that safety issues are addressed effectively. Details regarding software safety should be included in the System Safety Program Plan.

The SDP should require software safety engineers to define classifications for safety critical SIs and SUs. All SIs and SUs should be categorized according to these safety critical classifications. To prepare these classification levels, consideration should be given to: the severity and probability of hazards the SIs or SUs may contribute to (as determined by the Hazards Analysis); the potential for the SIs or SUs to provide safety-critical monitoring or mitigation actions; and how the SIs or SUs handle and protect safety critical data. System and software safety engineers should:

- Participate in system and software requirements analysis to generate additional functional or performance requirements to assure safe operations and safety contingency actions
- Monitor these additional software requirements to assure they are properly specified and traced to documented safety critical hazards
- Assure that unsafe operations are not specified by existing requirements
- Participate in design reviews to prevent unsafe approaches from being applied
- Track internal and external safety-related interfaces to assure they are fully documented and unambiguous
- Participate in the review of test procedures to assure safety critical requirements are properly interpreted and tested
- Participate in the evaluation of safety-critical code changes and review regression tests
- Document safety critical criteria used in selecting COTS, GOTS, and reuse code

#### 4.2.5.2 Software Information Assurance

Security requirements involve SIs and SUs whose failure may lead to a breach of system security or a compromise of classified data. Each software-related security-critical requirement identified should be documented in the Security and Privacy Protection Requirements section of the SRS and identified by a unique product identifier. Information Assurance (IA) requirements should be derived from the System Specification; IA concerns can have a significant impact on software architecture.

Security services provided by the program **must** be documented in the IA Plan and should provide “layers” of structured defense from commercial packages (such as anti-virus software and firewalls)

to elaborate National Security Agency (NSA) approved Type-1 encryption algorithms. The SDP **must** state that software subject to IA product certification and accreditation **must** be developed in accordance with the IA Plan. Software IA requirements should be flowed down through the normal requirements analysis process. The software design activity **must** conform to the IA architecture as described in the IA Plan. Also, when developing the software schedules, and the build plan, the IA product certification and accreditation need dates must be accounted for.

#### 4.2.5.3 Privacy Protection

Privacy-critical requirements are those requirements on SIs and SUs whose failure may lead to a compromise of private personal data such as training scores or personnel evaluations. Each software-related privacy-critical requirement identified should be documented in the Security and Privacy Protection Requirements section of the SRS and identified by a unique product identifier.

#### 4.2.5.4 Dependability, Reliability, Maintainability and Availability

Software plays a critical role in the overall dependability, reliability, maintainability, and availability of each segment. Mission Critical software (discussed in subparagraph 1.2.3.1) can be further defined as a software function that if not performed, performed out-of-sequence, or performed incorrectly, may directly or indirectly cause the mission to fail. The SDP should require a Failure Modes, Effects, and Critically Analysis (FMECA) for software to be performed for all new or modified mission critical software and require a list of SIs that are mission critical to be identified and maintained.

Dependability, reliability, maintainability, and availability all have quantitative as well as qualitative definitions. The qualitative and quantitative definitions are allocated to hardware and software from the higher level specifications. This section of the SDP should address the approaches to be used by software to ensure that both the qualitative and quantitative requirements are met.

**Dependability.** Dependability is the sum result of effective strategies for reliability, maintainability, and availability and the SDP should describe the overall approach proposed to develop these strategies. Software reliability and maintainability practices **must** be incorporated throughout all software development activities; they provide the building blocks for dependability and availability. Effective strategies for reliability and maintainability also help to ensure the software meets requirements with minimum risks, maintains the integrity of the software design, and minimizes lifecycle costs.

**Reliability.** Software reliability models should be used to assist in making predictions about the software system expected failure rates. The SDP **must** show that reliability tasks are integrated with quality assurance, product evaluations, maintainability, and other engineering activities to avoid duplication and provide a cost effective program. Software reliability should involve detection, reporting, quantification, and correction of software deficiencies throughout the software design, development and testing activities.

**Maintainability.** There are two aspects of software maintainability: software restoral and software repair:

- Software restoral is defined as the process of restoring the software to an operational state after a hardware or software failure has occurred. Software restoral can be a large contributor to downtime and thus can significantly affect system availability. The need for rapid software restoral is a major driver of the software architecture and design task.



- Development of maintainable software, from a software repair perspective, involves planning and establishing the software development methodology, environment, standards, and processes with an objective of making software maintenance changes efficiently and effectively.

Some methodologies, such as object-oriented design, development and programming, may produce software-related products that are more maintainable than other approaches. The design **must** be captured and retained in the software engineering tools and subject to configuration management (CM) processes. Similarly, the software CM tools provide support to software maintenance needs. Other tactics that can be described in the SDP to improve maintainability may include:

- The Software Engineering Environment (SEE), covered in SDP subsection 5.2, **must** be sized to include sufficient capacity to support post-deployment software support requirements, thus promoting long-term maintainability.
- Software standards **must** be established for each programming language to ensure that consistent programming styles are applied by all developers and that the software and supporting documentation are complete and understandable.
- The software product evaluations should assess compliance with the standards to ensure that they are consistently applied.
- Software change rates for units and functions may also be tracked as an indicator of more subtle maintainability factors.

**Availability.** A high availability rate for access to the system is the by-product of effective reliability and maintainability practices as well as accurate estimation of user needs. By performing modeling and trend analysis, based on historical trend data and collected metrics, software reliability and availability can be predicted and the necessary corrective actions can be taken to achieve the reliability and availability requirements.

#### 4.2.5.5 Assurance of Other Mission Critical Requirements

Critical software requirements should be tracked and monitored throughout the software development activities similar to other software requirements. However, in addition to the standard testing and quality assurance procedures for other software requirements, the Software IPT should follow an assurance strategy designed to ensure that hazardous or compromised conditions are eliminated or minimized for each development activity. This strategy should be to:

- Identify and document critical requirements in the appropriate SRS sections
- Document the specific SUs that contribute to the critical requirements through the traceability approach described in SDP paragraph 4.2.3
- Define specific SI testing procedures that execute all affected SUs to determine compliance
- Execute the security and privacy testing procedures at each SI build when affected security-critical and privacy-critical SUs have changed
- Execute the safety related test cases at each SI build for SIs with safety-critical SUs, even if the units have not changed
- Update safety analysis, models, and modeling results at any time

The CSWE and SQA should review the procedures followed by the Software IPT and the products produced for critical requirements compliance as part of the normal reviews of each development activity. The CSWE should focus on identifying evidence that the general strategy stated above is being implemented. SQA should evaluate the process of performing the critical requirements testing, the successful completion of the testing, and the proper documentation of the results.



#### 4.2.6 Computer Hardware Resource Utilization

Target computer hardware resource utilization **must** be recorded and monitored throughout the software development process by each segment for their respective computers. Resource utilization monitoring should be performed for all computers involved in the operational system. The Software Measurement (Metrics) Plan should define how computer hardware utilization is to be reported, and how the utilization data will be managed as Technical Performance Measures (TPMs).

The Integrated Product Teams (IPTs) should initially determine estimates of projected resource utilization measures. The measures may include memory utilization, processor throughput, input/output bandwidth, critical timing paths, and disk space or mass media storage. These measurements should be re-evaluated periodically during a build as actual utilization data becomes available. Since the software may only be partially completed at a build, the Software IPT analysts should extrapolate resource utilization.

#### 4.2.7 Recording Rationale for Key Technical Decisions

During the software development process, key technical decisions are made that may include specifying, designing, implementing, and testing SIs or SUs and issues related to interfaces, performance, functionality, etc. These decisions are usually captured in the resulting software, but not necessarily the rationale behind the decisions. Recording these decisions is important and frequently neglected.

The SDP should require the segments to identify key technical decisions as a natural and continual part of the development process. They should define those key decisions in the requirements definition, design, implementation, and testing activities that are determined to significantly impact the SI. The development teams should use their best engineering judgment. They can use the following subjective guidelines where an affirmative response indicates a key decision that needs to be recorded:

- Was a trade study, technical analysis, or software survey required to make the decision?
- Are there requirement, cost, or schedule factors that override the technical rationale for the particular decision made?
- Does critical rationale information exist that may be needed for future software maintainers?

When a key technical decision is identified, the rationale behind the decision should be documented in an Engineering Memo or meeting minutes and included, or referenced, in the SDF. The process is managed by Data Management, but the Software IPT **must** ensure all critical information is retained. The Software Item Leads should be responsible for ensuring compliance with this recording rationale.

#### 4.2.8 Access for Acquirer Review

The primary repositories for software products and related information are the Software Development Folders (SDFs) and Computer Assisted Software Engineering (CASE) tools. The work products produced during the development of the software **must** be kept under configuration control, both for configuration management and for customer review. The SDP **must stress full government access to the program's electronic website**. The following is an example of the contents for this paragraph:

**Example Text:**

In addition to unrestricted on-line access to documentation, the Program Office and their representatives participate in all joint technical and management reviews (as described in subsection 5.18). These reviews are held throughout the software development life cycle and consist of both formal contractual reviews and informal Technical Interchange Meetings. The Program Office, and their representatives, can also participate in the frequent telephone conferences and are also members of XMPL IPTs and the Software Engineering Process Group (SEPG).

**4.2.9 Software Data Management (Recommended Optional Addition)**

This is an optional, but highly recommended, additional paragraph of the SDP since software Data Management (DM) provides the interchange and access of controlled data to program personnel and the customer, supports timely delivery of contract deliverables, and addresses key issues such as disaster recovery and data rights. Software DM and the related concerns covering disaster recovery, proprietary rights, and international issues are not addressed in TOR-3537B or J-16.

The DM organization should be responsible for the repository and central access point for program and software documentation, the data accession list, storage media control, and informal documents. A DM Plan should detail the guidelines for preparation, identification, filing, retrieval, training, and standards for all program documentation. The DM Plan should be updated in accordance with evolving requirements of the contractual phases. The DM Plan is generally not a part of the SDP but there is no restriction preventing it from being an SDP addendum.

The Data Center is typically the hub of the DM task and the source for all configuration controlled documents including publication and distribution. Software documentation should be made available to the program team and the Government on the program's electronic website. Software development documentation **must** be retained in the Software Development Library (SDL) typically located at each development site.

**4.2.9.1 Disaster Recovery**

Plans for disaster recovery should be included in the SDP or in an external plan referenced by the SDP. Disaster recovery provides an alternate repository and backup system of software, databases, documentation, and equipment (if necessary). Disaster recovery plans provide an alternate development/operational capability in case of a catastrophic situation after initial delivery.

The disaster recovery plans ensure protection against loss of, or damage to, organizational assets and data. They ensure a smooth transition from normal to backup operations and ensure an expeditious restoration of the site capabilities.

**4.2.9.2 Proprietary Nature and Government Rights**

Rights restrictions apply as identified in the contractual Technical Data Restrictions. Vendor trademarked or copyrighted items **must** be used in accordance with applicable licenses; the Government **must** have the right to use these items in accordance with those applicable licenses. Restrictions on these tools (if any), other than those dictated by commercial practices, **must** be clearly described in the SDP and/or in the IMP. Proprietary concerns can be major issues in source selection. Data rights apply to all software products—not just code or COTS. This SDP paragraph needs to specify what standard level of data rights applies to each category of software and Software Item on the program.



### 4.2.9.3 International Traffic in Arms Regulations (ITAR)

If development leverages technology and products from foreign countries, ITAR is likely to apply and this issue **must** be addressed. The local ITAR Compliance office **must** be consulted for specifics if this is a program issue.

### 4.2.10 Software Plans and Work Products (Recommended Optional Addition)

This is an optional, but highly recommended, additional paragraph of the SDP. Software work products may include documentation, test results, non-document work products, and the source code itself. Minimum work products vary according to software category and, of course, the program's Contract Data Requirement List (CDRL). Section 5 of the SDP describes the detailed software development activities and its subsections contain a list of software work products produced during each activity.

#### 4.2.10.1 Software Management and Quality Control Plans

Addenda to the SDP, covering management plans and quality control plans, may be an integral part of the SDP or bound separately. In either case, these management and quality control plans represent important adjuncts to the SDP that document specific implementation details not covered in the main body of the SDP. Table 4.2.10.1 contains a list, and a brief description of the purpose, of the typical management and quality control plans that can be included as addenda to the SDP if they provide value added to the program.

Table 4.2.10.1. Candidate Software Management and Quality Control Plans—Example

Name of Plan	Purpose of Plan
<b>Software Metrics Plan or Guidebook</b>	Describes the approach, guidelines, and "how to" instructions for establishing a standard software metrics program across the software development effort. It contains specific user instructions as to what measurements to make, when to make them, calculations needed to translate the measurements into useful management data, analysis techniques, and report format examples.
<b>Software Subcontract Management Plan</b>	This plan may be included in the program's Subcontract Management Plan. It describes what software is subcontracted, and to whom, responsibilities of the Subcontract Management Team, identification of its members, responsibilities of the software subcontract technical manager, subcontract tracking and oversight of software activities, and references to contractual commitments.
<b>Software Risk Management (or Mitigation) Plan</b>	The plan for determining and mitigating software related risks. It describes the approach to identification and management of risks inherent in the development effort including reliability, design, cost, and schedule risks. It should assign a risk severity level to each identified risk, define risk handling plans where needed, the process for assuring implementation, and provide plans for maintaining and improving maturity levels of team members. It may be included in the program's Risk Management Plan.
<b>Software Data Management Plan</b>	Provides details on the scheduling, formatting, delivery, storage, and control for program deliverable and non-deliverable software documentation and media. It describes how the program provides: current program information; expedient interchange and access of controlled data to program personnel; timely delivery of contract deliverables; the repository and central access point for software documentation; the data accession list; document storage media control; and the focal point for software-related information. It <b>must</b> also include the mechanism for electronic access to the data by the customer.
<b>Software Reviews Plan</b>	Provides software management with the controls necessary to oversee software development review activities and provides software engineers with the standards and practices required to conduct software development reviews. It describes the objectives, frequency, and products reviewed, and establishes the entry and exit criteria for each review.



Name of Plan	Purpose of Plan
<b>Software COTS/Reuse Plan</b>	Covers COTS/Reuse software evaluation, selection, procurement, development environment requirements, special COTS/Reuse Configuration Management procedures, acceptance procedures, integration, and implementation, maintenance, evolution, and vendor monitoring and management.
<b>Software Resource Estimation Plan</b>	Describes the derivation of software resources needed and should include: software size; development effort; schedules and milestones; costs; and critical computer resources. It should describe the processes for: making estimates and periodic refinements with actual measurements; documenting results; and using parametric estimating models.
<b>Software Roles and Responsibilities</b>	Summarizes the roles and responsibilities for each software engineering skill group (usually in tabular form) including: IPT Lead, Chief Software Engineer, Segment Chief Software Engineer, Software Process Lead, IPT Software Lead, IPT Software Integration and Test Lead, Software Item Lead, software engineers, software test engineers, software configuration management.
<b>Software Safety Plan</b>	Describes the safety-critical safeguards that <b>must</b> be built into the software when human safety is involved. It may be incorporated into other documents such as the "System Safety Program Plan" or the "Risk Management Plan."
<b>Software Configuration Management Plan</b>	Establishes the plan for creating and maintaining a uniform system of configuration identification, control, status accounting, and audit for software and software work products throughout the software development process including the Corrective Action Process.
<b>Software Quality Assurance Plan (or Software Quality Program Plan)</b>	Establishes a planned and systematic software quality process to ensure that the software products and software processes comply with program contractual requirements as well as program process and product standards. It identifies the activities performed by the SQA organization in the development of all SIs and describes the SQA policies, procedures, and activities to be used by all software development team members.
<b>Software Quantitative Management Plan</b>	A high-level plan for establishing quantitative management on a program including quality goals, customer goals, other goals to supplement the IMP, priorities, and metric limits. It may be incorporated into the Software Metrics Plan or Guidebook.
<b>Software Process Improvement Plan</b>	Describes how process improvement is integrated into the management culture, and the plans for implementing a managed, iterative, and disciplined process for improving software quality, increasing productivity, reducing cost and schedule, and eliminating activities of little value. It should describe the controls, coordination and information feedback needed from: the software development process; the defect detection, removal and prevention process; the quality improvement process; and the software metrics program.
<b>Software Peer Review Plan</b>	Defines the procedures, data collection, responsibilities, and reporting needs for inspections and evaluations of software products.

#### 4.2.10.2 Detailed Software Work Instructions and Procedures

The defined software process, as captured in the SDP at a relatively high level, should be elaborated through detailed Work Instructions and/or Procedures. These instructions or procedures should contain detailed directions for the day-to-day implementation of the software process. A list of the Work Instructions and Procedures can be included in an SDP Appendix but the Work Instructions and Procedures themselves should be bound separately as they are typically voluminous. Table 8.3 in this Guidebook contains an example list of Work Instructions and Procedures.

Detailed procedures are often based on heritage or organizational sources for similar activities, customized for the program's use. For activities shared across each program, common procedures may be developed. The Software Engineering Process Group (SEPG) should maintain an inventory of approved software procedures for the program.

#### **4.2.10.3 Non-Document Software Work Products**

It is important to note that not all of the software work products are documents. The following are examples of software work products that may be produced during software development:

- Software requirements database
- Software Architecture diagrams/Data Flow Diagrams/Interface Design Diagrams
- Engineering Memos/Software use cases and scenarios/N-squared charts
- Simulation models and design captured in Object-Oriented (OO) models
- State Transition, Software Hierarchy, and Functional Block diagrams
- Management status reports and briefings
- Software productivity reports
- Design review packages





## 5. Detailed Requirements

This SDP section describes the activities, tasks, requirements, and responsibilities for developing the software. The development process described in this section is consistent with the software process defined in TOR-3537B. On a typical program, Section 5 should also be a tailored version of the developer's corporate Standard Software Process (SSP).

Table 5 contains a list of the sections comprising Section 5. The left half of Table 5 is a list of activities for the software development process and includes SDP subsections 5.3 through 5.13 and 5.26. The right half of Table 5 is a list of the activities that are "activity independent" as they support the entire software development lifecycle and includes SDP subsections 5.1, 5.2 and 5.14 through 5.25. Subsection 5.26 is optional but recommended if applicable.

Table 5. Contents of SDP Section 5

Subsection	Process Activities	Subsection	Independent Activities
5.3	System Requirements Analysis	5.1	Project Planning and Oversight
5.4	System Design	5.2	Establishing a Software Development Environment
5.5	Software Requirements Analysis	5.14	Software Configuration Management
5.6	Software Design	5.15	Software Peer Reviews and Product Evaluation
5.7	Software Implementation and Unit Testing	5.16	Software Quality Assurance
5.8	Software Unit Integration and Testing	5.17	Corrective Action
5.9	Software Item Qualification Testing	5.18	Joint Technical and Management Reviews
5.10	Software/Hardware Item Integration and Testing	5.19	Risk Management
5.11	System Qualification Testing	5.20	Software Management Indicators
5.12	Preparing for Software Transition to Operations	5.21	Security and Privacy
5.13	Preparing for Software Transition to Maintenance	5.22	Subcontractor Management
5.26	Software Sustainment (Optional)	5.23	Interface With Software IV&V Agents
		5.24	Coordination With Associate Developers
		5.25	Improvement of Project Processes

### 5.1 Project Planning and Oversight

The major objective of this first software activity is to complete and document initial planning for the software development task. The planning activity is an on-going task because it is initially performed as part of the draft SDP submission, with the proposal, and may be repeated several times with changing requirements of the program. This task is critical at the start of the development lifecycle as it is the foundation for producing the software plans required to implement and perform the software development process and for the identification and formation of the software teams required to execute those plans.

Software management has cognizance over the Software Development Plan (SDP) including the software management and quality control plans shown in Figure 1-2. The preparation of the Software Configuration Management Plan (SCMP), and the Software Quality Program Plan (SQPP) should be assigned to the Software Configuration Management (SCM) and the Software Quality Assurance (SQA) activities, respectively. The SDP contents **must** be consistent with Appendix-H of TOR-3537B and should be submitted to the contractor's Configuration Control Board (CCB) for approval before it is released for implementation. In accordance with TOR-3537B, the software Project Planning and Oversight activity **must** be described in six paragraphs in the SDP:

- Software Development Planning (paragraph 5.1.1)
- Software Item Test Planning (paragraph 5.1.2)
- System Test Planning (paragraph 5.1.3)
- Planning for Software Transition to Operations (paragraph 5.1.4)
- Planning for Software Transition to Maintenance (paragraph 5.1.5)
- Following and Updating Plans (paragraph 5.1.6)

A summary example of the readiness criteria for the Project Planning and Oversight activity, is shown in Table 5.1; it includes the entry and exit criteria, verification criteria to ensure completion of required tasks, and the measurements usually collected.

Table 5.1. Readiness Criteria: Project Planning and Oversight—Example

Entry Criteria	Exit Criteria
<ul style="list-style-type: none"> <li>• A management decision to initiate planning has been issued.</li> <li>• Customer system requirements are available.</li> <li>• IMP and IMS are available.</li> <li>• Software WBS has been defined down to the SI level.</li> <li>• Program Risk Management Plan has been established.</li> <li>• Software, Systems, and Project Teams are sufficiently formed to support the software planning activity.</li> </ul>	<ul style="list-style-type: none"> <li>• Software plans are placed in the electronic database.</li> <li>• Software size estimates are established; budgets and schedules are baselined.</li> <li>• SDP is reviewed and approved by all software team members and the customer.</li> </ul>
Verification Criteria	
<ul style="list-style-type: none"> <li>• Program software plans are reviewed and approved.</li> <li>• Program and senior management are provided status of ongoing product engineering activities (including requirements definition and management) on a periodic and event driven basis.</li> <li>• SQA performs process and product audits for the software planning activities per SDP Subsection 5.16.</li> </ul>	
Measurements	
<ul style="list-style-type: none"> <li>• Program schedule showing planning activities—estimated and actual.</li> <li>• Staffing levels planned versus actual</li> <li>• Effort hours bid, budgeted and actual</li> <li>• Milestone due dates—contractual, estimated, and actual</li> </ul>	
(see subsection 5.20)	

### 5.1.1 Software Development Planning

The SDP provides an important mechanism for documenting and tracking the software development effort and activities required by the software-related provisions of the contract. The program-level SDP **must** define software activities common to all development sites. Segment or Subsystem SDP Annexes (also called Site-Specific SDPs) can be produced containing specific and/or unique policies and procedures applicable to the segment/subsystem that expand on, but do not conflict with (except for approved waivers), the policies and procedures defined in the program-level SDP.

Software development planning information should be prepared by the Segment IPTs by augmenting the activities in the Integrated Master Plan (IMP) and the Integrated Master Schedule (IMS) with more detailed development schedules. Although these schedules may be in the SDP, it is generally better to reference their location since schedules typically change more often than the SDP is updated.



Once these schedules are complete, development oversight begins by monitoring products and processes and taking corrective action when necessary.

Unplanned updates to the SDP **must** be handled through the corrective action process described in SDP subsection 5.17. All changes to the SDP should require approval by the Chief Software Engineer (CSWE) and the program Software Engineering Process Group (SEPG). This SDP Guidebook assumes the program has a CSWE (see last paragraph of subsection 1.1). Changes to the SDP also should require CCB approval.

**Software Item Database.** A database, that may be called the Software Entity Database (SWED), should be produced and periodically updated, to provide a mechanism for identifying and profiling all SIs on the program. It provides a tracking mechanism for all software on the program. Each segment should be responsible for their data input to the SWED but the CSWE should be responsible for compiling this information into a single centralized and controlled database for the program. This database may include for each SI in the program: a functional description, class, category, size, percent of new versus reused code, responsible developer(s) and contact information, and language(s) used.

**Waiver Processing.** When a software development site has a justifiable reason for not complying with a required procedure in the SDP, they **must** submit a request for a waiver in accordance with the waiver process described in the SDP. All waivers should require the pre-approval of the CSWE and the SEPG prior to being sent to the CCB for approval. Figure 5.1.1 is an example of a waiver processing process.

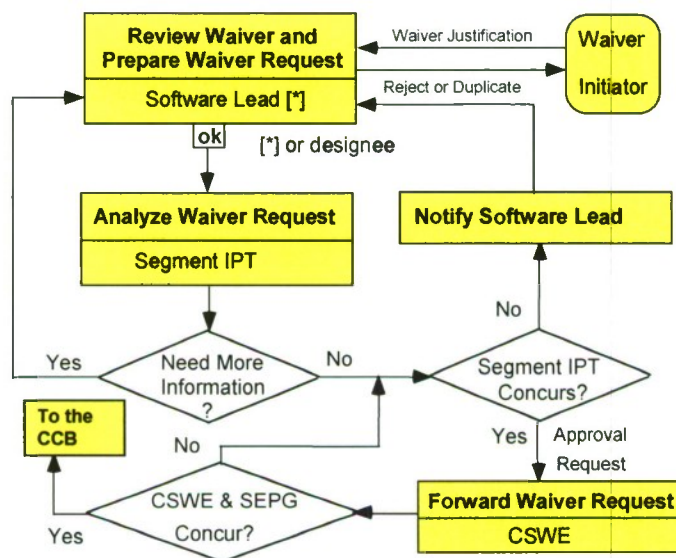


Figure 5.1.1. SDP Waiver Approval Process—Example

When a waiver is initiated, the justification for it should be presented to the Software Lead. If the development team Software Lead (or designee) agrees with the need for a waiver, a Waiver Request should be prepared requesting authority to deviate from the requirements in the SDP. The Software Lead should forward the Waiver Request to the Segment IPT for approval. The Segment IPT should review the waiver request, request more information or clarification if necessary, and either approve or deny the waiver request.



If the Waiver Request is approved, the IPT should forward it to the CSWE. If not approved or found to be a duplicate, it should be returned to the Software Lead with reason(s) for disapproval. If the CSWE and SEPG concur with the Waiver Request it should then be forwarded to the CCB for formal approval.

#### 5.1.1.1 Software Development Planning Tasks

Software planning is iterative and should not start until the assignment of planning roles has been made. Software planning responsibilities normally resides with the IPT software leads for development implementation planning and the program-level SEPG for process planning. The first step in planning is to review software requirements (see subsections 5.3 and 5.5) since the scope of the software task is established by identifying system requirements to be satisfied by software products. Table 5.1.1.1 is an example list of typical planning activities as elaborated in the identified sections of the SDP.

Table 5.1.1.1. Software Planning Tasks—Example

Software Planning Tasks	Covered In SDP Section/Subsection
Methods for developing and maintaining the SDP	1.3.3 SDP Updates
Software Data Management (DM)	4.2.9 Software DM
Software size and resource estimation	5.1.1.2 and the Software Estimation Plan
Software build planning	5.1.2 and the Software Build Plan
Software integration and test (I&T) planning	5.1.2 through 5.1.5 and the Software IT&V Plan
Software Development Environment and support tools	5.2 Establishing a SDE
Software acceptance, delivery, installation, transition, operations, maintenance, and retirement planning	5.12, 5.13, and the Software Maintenance and Transition Plans
Software Configuration Management (SCM)	5.14 and the SCM Plan
Software evaluations with formal and informal reviews	5.15, 5.18, and the Software Reviews Plan
Software Quality Assurance (SQA)	5.16 SQA
Problem resolution methods and preventive action	5.17 Corrective Action
Software risk management	5.19 and the Software Risk Management Plan
Software metrics covering products and processes	5.20 and the Software Metrics Plan
Security and Privacy issues	5.21 Security and Privacy
Oversight of software subcontracts	5.22 and Software Subcontract Plan
Software schedules with critical interdependencies	6 Schedules and Activity Network
Software organization, roles, and responsibilities	7.1 Project Organization
Required resources, skills and staffing plan	7.2 Project Resources
Training plans and training requirements	7.3 Training Plans
Software Operations and Maintenance	5.26 Software Sustainment

#### 5.1.1.2 Software Resource Estimating

Software resources, including physical, personnel, cost and computer resources, **must** be estimated before software development can begin. These estimates are used to establish software development schedules, risk mitigation plans, and commitments and should be documented in a Software Resource Estimation Plan.

Software personnel should participate with other affected groups (systems engineering, SQA, SCM, test, etc.) in the overall program planning throughout the program as members of Integrated Product Teams (IPTs). Commitments, or changes to commitments, made to individuals and external groups **must** be reviewed with management regularly.

**Staffing Estimation.** To determine the level of staffing required, the planning function should consider program constraints including milestones, reviews, documentation deliveries, product

deliveries, internal milestones, incremental builds, technical constraints, and any changes in scope. Estimates of source lines of code and software development productivity play an important role in staffing estimates.

**Re-planning:** The software groups should participate, when required, in re-planning activities to address contract changes, process improvements, or when measured performance varies from planned performance. The related data that is generated **must** be maintained and placed in the applicable Software Development Folders (SDF) or Software Engineering Notebooks. Software personnel also should participate in contract/subcontract modification activities (such as engineering change proposals).

### 5.1.1.3 Software Build Planning

A software “build” is a portion of a system that satisfies, in part or completely, an identifiable subset of the total end-item or system requirements and functions. There may be multiple internal builds leading to a deliverable build for an increment in the lifecycle. Requirements met in one incremental build are also met in all successive increments. The final build is the complete software system. A “release” is a build version that is delivered for acceptance testing and subsequently may be released or delivered for operational use. Incremental builds can be planned for each SI, or group of SIs.

**Build Requirements.** The Software systems engineering function should define the level of requirements satisfaction needed by each lifecycle increment to implement a specified level of end-to-end system functionality. Within a segment, additional influences dictate when capabilities are delivered. This may include such factors as developing required software infrastructure or addressing areas of high-complexity. Naming conventions for each build **must** be established up front by assigning unique alphanumeric designations.

**Software Build Plan.** A table, similar to the example in Table 5.1.1.3, **must** be added either to subparagraph 5.1.1.3 or, if too long, included in the SDP Appendix or a separate document referenced by the SDP, to show the intended software delivery plan. **The table must include a unique number, often called the Program Unique Identifier (PUI), for each Software Item and its name, the responsible developing organization, and Equivalent Source Lines of Code (ESLOC) planned for each build.** Part 2 subparagraph 1.2.3.3 provides an explanation of how ESLOC is derived. As shown in Table 5.1.1.3, the version (preliminary, initial, update, fixes) can be identified for each delivery.

Table 5.1.1.3. SI Build Delivery Plan-Example

PUI	Software Item Name	Developer	Build 1	Build 2	Build 3	Total
Total for 18 SIs:			102,000	65,000	130,000	297,000
1	Decision Support Manager		45,000	28,000	48,000	121,000
1.1	Decision Analysis	Able Corp	I	UDRW	U	
1.2	Analytical Algorithms	Able Corp	–	I	U	
1.3	Scenario Analysis	Able Corp	–	I	U	
1.4	Testbed Controls	Baker Co.	P	I	U	
1.5	Traffic Control	Baker Co.	I	UDRW	U	
1.6	Simulation Analysis	Baker Co.	P	I	U	
2	Services Support Manager		30,000	18,000	12,000	60,000
2.1	Routing Analysis	Charlie Co.	P	I	U	
2.2	User Support	Charlie Co.	–	I	UDRW	
2.3	XYZ Services	Charlie Co.	I	UDRW	UDRW	
	Etc.	Etc.				

P = Preliminary Version

U = Updated Delivery

UDRW = Updates for Discrepancy Report Work-Offs

PUI = Program Unique Identifier

I = Initial Delivery

– = No Delivery

##,### = ESLOC per Build



**Software Master Build Plan (SMBP).** A comprehensive SMBP **must** be provided to map the incremental functionality, capabilities, and requirements allocated to each build. The CSWE, or Build Manager, usually maintains the SMBP with the approval of the software CCB. Once approved, the SMBP should be controlled by SCM and the CSWE, or Software Build Manager, should routinely report the status and changes to program management. The SMBP may also be called the “Master Software Integration and Verification Plan” or may be referred to as a “Build Functionality Matrix.”

**Build Planning Updates.** Software build planning should occur for each program increment and each deliverable build and be updated continuously throughout the program. Build plans are typically updated only when the plan contents change significantly as determined by the IPT Lead. Schedule, ESLOC, and functional content estimates **must** be taken into consideration when planning builds. As the program matures, additional design, requirements, technical content, and testing approaches should be added. The build activities should be documented in detailed schedules and then incorporated into the IMS along with staffing and budget-plan information.

#### 5.1.1.4 Software Development Tracking and Oversight

The software tracking and oversight effort begins once software planning is complete. Segment IPTs, Leads, the CSWE, and SQA monitor software development status by:

- Collecting and evaluating software metrics data (SDP subsection 5.20)
- Evaluating software products (SDP subsection 5.15)
- Performing product quality and process audits (SDP subsection 5.16)
- Supporting software reviews (SDP subsection 5.18)
- Performing risk management activities (SDP subsection 5.19)

**Software Measurement Oversight.** Throughout the development process, software measurement data **must** be used to compare actual software size, cost, schedule, and progress against the established plan (see subsection 5.20). If the metrics indicate out-of-tolerance conditions, the segment IPT software members perform an analysis to determine the corrective action and potential risks including cost and schedule impacts. **The Software Measurement (or Metrics) Plan is an important addendum to the SDP.**

Software measurement data reported to project management **must** also be reported to the customer and corporate senior management. The status of software should be reviewed weekly at segment level meetings and at monthly program status meetings. In addition, software status should be provided to the customer monthly and also at quarterly reviews. Software management and control **must** be integrated into the overall program management scheme. Figure 5.1.1.4 is a depiction of software management from a measurement perspective.

**Cost Account Oversight.** Software work packages are typically cost accounts within the Earned Value Management System (EVMS) used by all contractors. The cost account **must** be at a level of detail sufficient to maintain control of the associated software development activities. Metrics on the cost accounts/work packages should be reported to program management and available to the customer.

**Schedule Oversight.** Schedule review meetings should be conducted weekly. Schedule metrics (using weekly milestone accomplishments, including subcontractor data) should be reported along with status of corrective action/recovery plans. IMS and detailed schedules should also be reviewed at lower levels within the IPTs.



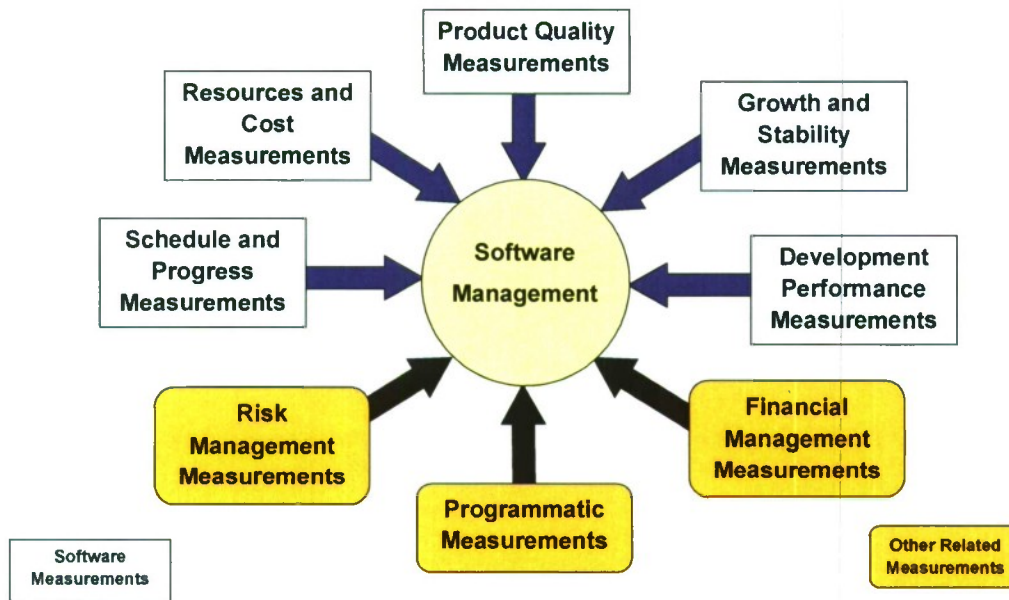


Figure 5.1.1.4. Software Management from a Measurement Perspective—Example

**Headcount Oversight.** IPTs should monitor headcount on a weekly basis and strive to identify potential problems early. Updates of accomplishments, actual headcount, budget and forecast should be conducted on a monthly basis. Forecasts should be updated and reported in internal cost performance reports that include significant cost/schedule variances and changes in the latest revised estimate. Costs and schedules should be controlled by monitoring headcount and expenditures, and by assessing progress.

**Product and Process Oversight.** Product evaluations, software reviews, process audits, and assessments are used by segment IPTs, CSWE, and SQA as a means to determine compliance with the standards established by the SDP. Non-compliance of baselined products is handled via the corrective action process (CAP) (see subsection 5.17). Process audits **must** be performed by SQA, with support from the CSWE, to determine compliance with the processes specified in the SDP (see subsection 5.16). SQA **must** be responsible for documenting and verifying closure of a non-compliance issue. Subsection 5.15 describes the software product evaluation process. Software management **must** implement and maintain the mechanisms for interfacing to and communicating with the customer.

## 5.1.2 Software Item Test Planning

This paragraph of the SDP **must** contain the approach for performing the Software Item Test Planning. The testing of segment SIs **must** be performed by the respective segment software test engineers. They are responsible for documenting the Software Test Plan (STP), Software Test Description (STD), and Software Test Report (STR) to verify that the SIs meet the allocated requirements.

A preliminary version of the STP is usually produced during the software design activity (see SDP subsection 5.6). However, the STP is the result of the SI test planning activity. Data Item Description DI-IPSC-81438A, or Annex E.2.2, in J-16, should be used as a guide for preparing the STP. Production of the STD and STR is performed during Software Item Qualification Testing (SIQT) and is discussed in SDP subsection 5.9. Test activities for MC and SS software classes **must** also be documented in the SDFs.

The STP describes plans for qualification testing of SIs and is an important software document. It describes the test environment to be used, identifies the tests to be performed, provides schedules for the testing tasks, defines the resources needed, and addresses all of the planning tasks required to conduct the SIQT. Table 5.1.2 summarizes the readiness criteria in terms of entry, exit and verification criteria to ensure completeness of the STP.

Table 5.1.2. Readiness Criteria: Software Test Plan—Example

Entry Criteria	Exit Criteria
<ul style="list-style-type: none"> <li>• The appropriate STP DID and other reference materials are obtained.</li> <li>• Software requirements are established in the SRS and IRS and are traceable to a parent requirement.</li> <li>• The top-level software architecture is established.</li> <li>• The Verification Cross Reference Matrix (VCRM) specifies the test verification method and level for each requirement in the SRS and IRS.</li> </ul>	<p>The following tasks have been defined and documented in the STP:</p> <ul style="list-style-type: none"> <li>• Test environment (sites, hardware, software, test tools, test facilities, test data, etc.) needed to conduct the lifecycle tests.</li> <li>• Test scenarios to be performed including the schedule for executing the test activities.</li> <li>• Traceability between SI requirements and the related tests and test phases where the requirements are verified.</li> <li>• Personnel, organizations, responsibilities, and management activities needed to develop and implement the planned testing.</li> <li>• The objectives for each test including test level, type, conditions, data to be recorded, qualification method(s), data analysis, assumptions and constraints, safety, security, and privacy considerations.</li> <li>• Approach to related issues such as data rights, training, regression testing, delayed functionality, and deliverable documentation.</li> <li>• Criteria for evaluating the test results.</li> </ul>
Verification Criteria	
<ul style="list-style-type: none"> <li>• The tests identified fully test and verify the requirements being tested.</li> <li>• The occurrence and timing of the test phases in the lifecycle, plus the entrance and exit criteria for each test phase, has been identified and documented.</li> <li>• The terminology and format is consistent between the SRS, RTVM, IRS, and STP.</li> <li>• The STP has successfully passed its peer review.</li> </ul>	
Measurements	
<ul style="list-style-type: none"> <li>• Statistics from the STP peer review.</li> </ul>	

### 5.1.3 System Test Planning

This paragraph of the SDP **must** contain the approach for providing support to system test planning. The System Verification and Test Plan (it may also be referred to as an Integration, Test, and Evaluation Plan) should be prepared by the SEIT and is the key planning document for system testing. System integration and test activities are described in SDP subsection 5.11.

The System Test Team should be responsible for performing the actual system testing. Software developers and/or software test engineers have a support role in system test planning that may include reviewing test preparation materials, and providing software test support items, such as reusable software test documentation, simulators, drivers, and analysis tools. Software engineers also support anomaly analysis to determine if the problem is because of software only, hardware only, or a combination. If regression testing on the software builds is needed, SCM **must** provide the software builds against which the tests are conducted.

### 5.1.4 Planning for Software Transition to Operations

This paragraph of the SDP **must** contain the approach for performing the software installation planning. This activity involves the preparation for, and the installation and checkout of, the executable software at a user site. As described in SDP subsection 5.12, planning and preparation should start relatively early in the lifecycle to ensure a smooth transition to the user. It should include the preparation of documentation and software products required by the user to perform operational tasks. This includes the code for each SI and supporting documentation including the preparation of user manuals and user training materials as the pertinent information becomes available. Annex E.2.3,



in J-16, should be used as a guide for preparing the Software Installation Plan (SIP). SDP subsection 5.26 discusses Software Sustainment issues.

### 5.1.5 Planning for Software Transition to Maintenance

This paragraph of the SDP **must** contain the approach for performing the software transition planning. As described in SDP subsection 5.13, transition planning involves advance planning and preparation that should start early in the lifecycle to ensure a smooth transition to the maintenance organization. It **must** include the installation and checkout of the software at the maintenance site. Either DID DI-IPSC-81429A or Annex E.2.4, in J-16, can be used as a guide for preparing the Software Transition Plan (STrP). SDP subsection 5.26 discusses Software Sustainment issues.

### 5.1.6 Following and Updating Plans

The plans noted in paragraphs 5.1.1 through 5.1.5 **must** be made available via an electronic data access system accessible to all stakeholders and the customer. Once baselined, unplanned modifications to these plans should be made via the corrective action process. Modifications that are planned, such as scheduled updates to baselined documents at major milestones **must** also be electronically available. Unplanned modifications, may also be captured as lessons learned. This section should also cover the contractor's approach to enforcement of planned updates to the plans.

The SEPG should review the software development process at monthly SEPG meetings to determine the effectiveness of the process through analysis of software metrics, requests from Segment IPTs, recommendations from SEPG members, the customer and their representatives, and process audit information from SQA, and program directives.

If other software or program level plans are affected by the approved change to the SDP, the CSWE **must** ensure that responsible parties are notified of the SDP update and ensure that all inter-group commitment changes are coordinated. The SEPG, described in paragraph 5.25.1, should coordinate this activity. Unplanned changes to this SDP **must** be initiated and tracked using the corrective action process described in subsection 5.17.

## 5.2 Establishing a Software Development Environment

A Software Development Environment (SDE) **must** be established to meet project software development and test requirements. In accordance with TOR-3537B, the SDE activity **must** be described in five paragraphs in the SDP:

- Software Engineering Environment (paragraph 5.2.1)
- Software Integration and Test Environment (paragraph 5.2.2)
- Software Development Library (paragraph 5.2.3)
- Software Development Files (paragraph 5.2.4)
- Non-Deliverable Software (paragraph 5.2.5)

### 5.2.1 Software Engineering Environment

The Software Engineering Environment (SEE) **must** consist of the hardware, software, procedures, and documentation necessary to support the software development effort. Core Computer Assisted Software Engineering (CASE) tools used across the program **must** be identified in a table similar to Table 5.2.1-1. The mechanism for making changes to the program-wide SEE CASE tool set should be by approval of the Software CCB. Additional SEE requirements **must** be defined—typically in segment/subsystem SDP Annexes, in a similar table.



Table 5.2.1-1. Program-wide SEE CASE Tools—Example

Purpose of Case Tool	Name of Tool	Vendor
Object-Oriented Analysis and Design	Rose 2000	Rational
Code Development and Testing	SparcWorks	Sun Microsystems
Large Relational Database	Oracle	Oracle
Small Relational Database	Access	Microsoft
Problem Tracking Reports	ClearQuest*	Rational
Planning and Scheduling	Project	Microsoft
Configuration Management	ClearCase*	Rational
Requirements Management	DOORS*	Telelogic
Software Estimation	SEER – SIM	Galorath
Software Metrics	DataDrill*	Distributive Software

\*Core software management tools used across the program.

Note: Use of trade names in this material is not intended in any way to infringe on the right of the trademark holder.

The example Table 5.2.1-1 can be expanded to include all tools and the segments/subsystems using each tool. This table can become lengthy (e.g., 100 tools) in large programs so it may be put in the SDP appendix. If it is lengthy, it is recommended that the tools be grouped in categories such as: Operating Systems, Compilers, Configuration/Change Management, CASE Tools, Requirements Traceability, Documentation, Metrics Collection and Analysis, Performance Analysis, and Test.

When CASE tools are selected for the program, it is important to remember that new tools are not likely to make an ineffective process more effective; new tools are not a panacea for fixing problems—but they can make an effective process more efficient.

The CSWE **must** coordinate implementation of the common tool suite among all segments/subsystems to ensure effective information transmittal and maximum commonality. The CSWE, or designee, should be responsible for monitoring the implementation of the SEE to ensure that all requirements are implemented, for periodically assessing the continuing adequacy of the environment, and for identifying additional needed tools. Details of the SEE configuration for each segment should be maintained in a current inventory list and available from the System Administrator at each site or segment.

An overview of the data network **must** be included as a figure in the SDP or referenced to its location. In addition to a data network diagram, the major operational software development sites should be listed in an overview table similar to Table 5.2.1-2. This table could incorporate an SI column, however, SIs are often developed at multiple sites.

Table 5.2.1-2. SEE Development Sites—Example

Location	Function	Function Name
City, State	FSS	Flight Software Subsystem
City, State	TT&C	Telemetry, Tracking, and Command
City, State	MPS	Mission Processing and Services
	FSE	Field Station Element
City, State	MMC	Mission Management Center

## 5.2.2 Software Integration and Test Environment

Each software development segment/subsystem (site or factory) **must** have a controlled test environment that supports integration and test of its SIs as part of its integrated SDE. These test

environments should be defined by segment/subsystem test personnel and described in their Software Test Plan (STP) and their SDP Annexes.

Care **must** be taken at all levels, including system integration, to procure the needed integration and test tools far enough in advance to assure they are available when needed and that there is enough time for user training. Some SIs may be developed at multiple sites. All software developed at geographically dispersed sites **must** be fully tested at each development site, preferably on target hardware, prior to final installation and qualification testing at the integration location.

All planned Integration and Test Environments should support testing using “Test-Like-You-Fly” principles. This includes high fidelity simulators and target test beds and test facilities that are representative of the operational environments.

### 5.2.3 Software Development Libraries

Two levels of software libraries are normally used to implement software CM as follows:

- The Master Software Development Library (MSDL) is a single master program-level repository of software information.
- Each software development segment/subsystem (or site) should maintain a subordinate Software Development Library (SDL) at its site for local control of software products.

These libraries **must** provide repositories for products resulting from software requirements definition, design, implementation, and test in accordance with the requirements of the SDP. The MSDL and SDLs **must** be controlled collections of software, documentation, and associated tools and procedures used in the development of software. The SDL for each development site **must** be defined in its respective SDP Annex. The MSDL and SDLs **must** be maintained throughout the contract duration. Also, electronic items **must** be maintained in a restricted environment and access controlled by login procedures.

The SDL contains code, test cases, and the electronic version of the software documentation. Figure 5.2.3 is an example of a typical logical partitioning of the SDL in electronic form. In Figure 5.2.3 the SDL is shown as three primary logical partitions: the software development area; the controlled library area; and the verification area as shown in the figure.



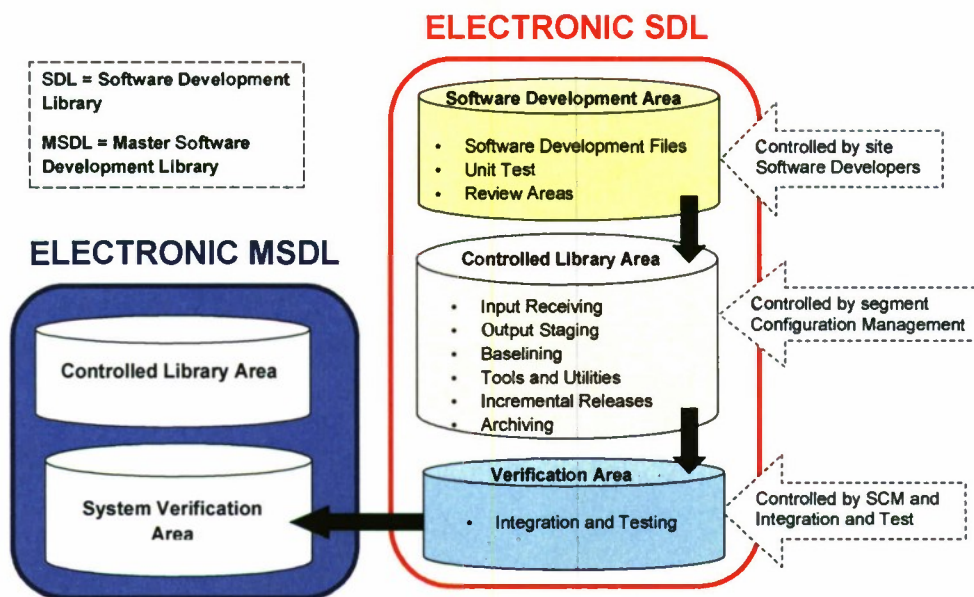


Figure 5.2.3. Electronic SDL Logical Partitioning—Example

The following paragraphs may be used as sample text describing the three electronic library areas including ownership and control of the partitions. It is intended to be general guidance as the specific organization of segment SDLs should be defined and described in the segment annexes.

**Example Text:**

**The Software Development Work Area** is maintained and controlled by the software engineers as a working area to develop the software. This working area is used to create new code and/or documents, modify previously released code and/or documents, maintain databases, perform unit testing by the software developers, and for other users to review the products electronically. At the end of each build, the finished products, and the SDFs, are transferred from the Work Area to the SDL Controlled Library Area.

**SDL Controlled Library Area.** After code and unit test is complete, the software goes under segment Configuration Management (CM) control (i.e., it is baselined). The CM group copies finished products received from the software developers into this area. CM will always rebuild the executables from the source files before transferring them. CM has ownership, full accountability, and full access privileges; all other users have *read-only* privileges in this area. Software products are held here for CM to verify that the necessary files have been received by doing a preliminary validation of the product. When all files are received, the executable software products will be built from the source code and transferred to the SDL Verification Area.

**The SDL Verification Area** is owned, maintained, and controlled by the integration and testing group. They have write privileges and no other users may modify the data in this area. Before products are transferred to the Verification Area from the Controlled Area, CM verifies that the product is complete, and is ready for integration with other parts of the system; when that is done, the code and executables are copied into this area. All software products promoted into the SDL Verification Area are under the strict control of the chosen configuration management tool. Following software CCB approval, files are transferred to the Verification Area in the MSDL.

### 5.2.3.1 Electronic Data Interchange Network

The SDP **must** describe the program's ability to provide continuously available, secure, encrypted remote access such that any authorized individual can view all data (documents, analysis, databases, or other information) using a standard web browser. The data **must** be safeguarded at multiple levels (i.e., Unclassified, Contractor Proprietary and Secret levels) in accordance with Government



requirements and negotiated restrictions to rights in technical data and software. Access **must** include data generated by contractors and all subcontractors.

### 5.2.3.2 Software Process Assets Repository

In addition to the MSDL and SDLs, software documentation should be provided, via electronic access, from a program-level library that may be called the Software Process Assets Repository (SPAR). The SEPG should be responsible for defining and maintaining the SPAR; this repository usually consists of both electronic and non-electronic materials.

### 5.2.4 Software Development Files

Software Development Files (SDF) are required for all software categories at the segment, SI, and SU levels. SDFs **must** be prepared and kept current throughout the program duration. If a SI or SU is deleted, its data **must** be retained in an inactive file. Since SDFs are involved throughout the software development process, subsections 5.3 through 5.11 of the SDP do not always call out the use of SDFs. The ubiquitous nature of the SDF should be understood when reading these sections.

**SDF Audits.** SDFs should be inspected and audited throughout the program, to determine compliance with the SDP, with at least one inspection performed during each build and prior to each major review. Deficiencies identified during these inspections normally result in corrective action through the corrective action system. The frequency of SDF audits should be defined in the Software Quality Program Plan (SQPP). After an SDF inspection by SQA, the CSWE, or the customer, the SDF **must** be updated to note that it has been audited.

**SDF Format.** SDFs can be maintained either in electronic format or non-electronic format for hard copies. Electronic information should be the preferred format. Information can either be placed directly into the SDF or provided by pointers to an external location. SDFs should be initiated during software requirements definition and remain under control of the segment/subsystem development teams from the time they are created until completion of the contract. Table 5.2.4 is an example tabular version of the overall SDF organization.

Table 5.2.4. Electronic SDF Organization—Example

SDF Folder	Description
SI Name	Root folder for each Software Item
Referenced locations	File(s) with pointers or links to SDF related materials in another location (to avoid duplication)
SI Level Peer Reviews	Peer Review materials (checklists, forms, notification; materials; log)
SU Level Peer Reviews	Peer Review materials (checklists, forms, notification; materials; log) for products for a specific SU
SQA Reports	SQA Audit support materials and reports
Assessment Reports	Assessment support materials, reports and management information
Lessons Learned	Lessons learned support materials and reports/meeting minutes/action items
Meeting minutes	Minutes not already stored in another location
Design materials	In-work design specifications; design definition; agreements/decisions; database from CASE tools
Plans, tracking, and support	Administrative, risks, low level schedules; trade studies and evaluations; BOE; prototype plans
Test artifacts	In-work plans, procedures; low-level (unit test; unit integration) plans, procedures, results, reports
Reviews and Presentations	Formal briefing and presentation materials and minutes
Training materials	Training and orientation materials and records not already recorded in another location.
Process Improvement	Plans, minutes, and reports
Metrics	Data, analysis and reports not already stored in another location
Tools	Common scripts and tools used that are not stored in another location

### 5.2.5 Non-Deliverable Software

Non-deliverable software consists of software developed, purchased, or used for software development but not required by the contract to be delivered to the acquirer or other designated recipient. It is identified as Category SS-3 in subparagraph 1.2.3.2. Non-deliverable software can be used during software development only if the operation and maintenance of the deliverable software does not depend on use of the non-deliverable software or the acquirer either has the software or can readily obtain it. In any case, the developer **must** ensure that all non-deliverable software performs its intended functions.

### 5.3 System/Segment Requirements Analysis

The major objective of this activity is the analysis and specification of system requirements. The activities in this activity are also equally applicable to Segment Requirement Analysis or any other level of requirements above software. The principal tasks performed in this activity should be led by the Systems Engineering Integration and Test (SEIT) organization with support from the segment Software IPT members.

In accordance with TOR-3537B, the System/Segment Requirements Analysis activity **must** be described in three paragraphs in the SDP:

- Analysis of User Input (paragraph 5.3.1)
- Operational Concept (paragraph 5.3.2)
- System/Segment Requirements (paragraph 5.3.3)

This activity is based on inputs from the customer and user-provided requirements such as: the Initial Capabilities Document (ICD), Capabilities Development Document (CDD), the Technical Requirements Document (TRD), the Statement of Objectives (SOO), and the Request for Proposal (RFP). The major output documents resulting from this activity are preliminary versions of the System/Subsystem Specifications (SSS), the Operational Concepts Description (OCD), and the Interface Specification (IS). The system verification and system test plans may also be revisited and updated if necessary.

In addition, interface definitions must be provided to enable the further definition and management of the computer software and computer equipment resources. This **must** be documented in the Interface Control Documents (IFCD). **The acronym 'IFCD' is used in this Guidebook to avoid confusion with the ICD defined in the above paragraph.** Depending on contract provisions, interface definitions may also be included in the System/Subsystem Specification (SSS) or the Interface Requirements Specification (IRS). The IRS may be contained within the SRS.

Inputs can also be derived from systems engineering studies. An early draft version of the SSS may also be provided to the contractors by the acquisition program office. Table 5.3 summarizes the readiness criteria for this activity with the entry and exit criteria, verification criteria to ensure completion of the required tasks, and the measurements usually collected.



Table 5.3. Readiness Criteria: System/Segment Requirements Analysis—Example

Entry Criteria	Exit Criteria
<ul style="list-style-type: none"> <li>External system interfaces have been identified and the related documentation has been reviewed.</li> <li>Preliminary concept of operations and system capability definition have been completed.</li> <li>Systems engineering notifies software team of the need for their support.</li> </ul>	<ul style="list-style-type: none"> <li>System level requirements analysis and segment requirements analysis is complete.</li> <li>Performance allocation, interface requirements, and user interface analysis are documented.</li> <li>System requirements joint technical and management reviews are successfully completed.</li> <li>Software representatives have reviewed system requirements and the concept of operations.</li> <li>System/segment requirements are allocated to software.</li> <li>Bi-directional traceability is completed from customer requirements to/from system specification and from system specification to/from segment specification.</li> </ul>
Verification Criteria	
<ul style="list-style-type: none"> <li>Software IPT personnel participate in the review and approval of the system and segment requirements and interface requirements documentation.</li> <li>Program and senior management are provided status of ongoing product engineering tasks (including Segment requirements analysis and management) on a periodic and event driven basis.</li> <li>System Requirements Review (SRR) is successfully completed.</li> </ul>	
Measurements	
<ul style="list-style-type: none"> <li>Requirements analysis task schedule.</li> <li>Number of system/segment requirements allocated to software.</li> <li>Planned versus actual level of effort.</li> <li>Requirements traced versus untraced.</li> </ul>	
(see subsection 5.20)	

### 5.3.1 Analysis of User Input

**System Requirements.** The SEIT has primary responsibility for the system-level tasks performed during this activity. The CSWE and/or the Chief Software Architect are often (and should be) part of the SEIT team. They directly support these tasks so that: (a) decisions involving software can be made with the appropriate expertise and (b) interface requirements are consistent across the system.

Segment Software IPT personnel support the system requirements analysis to ensure that requirements involving software are adequately addressed. The segment Software IPT assists and supports the SEIT in the identification and capture of the software needs by participating in system-level working groups.

The developers **must** also participate in analyzing user input to ensure that all interested parties maintain ongoing communications regarding user needs throughout development of the system. In addition to the developers, interested parties may include the users, acquirer, test, and maintenance organizations. Work products of this task may include need statements, surveys, SCRs/SDRs, the results of prototypes, and documented interviews.

**Segment Requirements:** Segment requirements analysis **must** be accomplished by analyzing allocated segment requirements from the system specification and interface requirements. Segment system engineering has primary responsibility for the segment tasks performed during this activity. The segment Software IPT should assist and support segment system engineering in the derivation of segment-specific requirements from the system-level requirements by participating in their working groups.

### 5.3.2 Operational Concept

The segment Software IPT should support the SEIT in defining the system Operational Concept Description (OCD) by identifying and evaluating alternative concepts for technical feasibility, user input, cost effectiveness, schedule impact, risk reduction, and critical technology limitations. The



segment Software IPT should also: (a) analyze the operational concepts and other inputs to derive any software requirements that are not specifically stated and (b) support the refinement of the operational concept based on current analyses and update it with user interface analysis material as appropriate.

### 5.3.3 System/Segment Requirements

System-level requirements **must** be documented in the SSS. This document specifies system capabilities and allocates requirements to the segments. Segment-to-segment, and system-to-external system interface requirements **must** be defined and documented during this activity.

All system, segment-segment, and segment-external requirements and interfaces should be maintained in a Requirements Database (see paragraph 4.2.3 for example requirements management databases). Segment Software IPT personnel should participate in working group discussions and joint IPTs to review and comment on the parent specification requirements related to software.

The segment Software IPT **must** support segment requirements analysis through the identification and derivation of software-related aspects for functional performance, interfaces, constraints, and quality requirements. These requirements **must** be analyzed for completeness, consistency, verifiability, and feasibility. Segment Software IPT participants also **must** identify and recommend requirements that could be feasibly allocated to and implemented in software and identify possible software verification methods and traceability for the segment requirements.

## 5.4 System/Segment Design

Segment Software IPT personnel **must** support the SEIT in developing the system/segment design and the specific configuration of hardware, software, and firmware to meet performance and reliability requirements. In accordance with TOR-3537B, the System/Segment Design activity **must** be described in two paragraphs in the SDP:

- System-wide/Segment-wide Design Decisions (paragraph 5.4.1)
- System/Segment Architectural Design (paragraph 5.4.2)

Table 5.4-1 summarizes the readiness criteria for this activity in terms of the entry and exit criteria, verification criteria to ensure completion, and the measurements usually collected.

Table 5.4-1. Readiness Criteria: System/Segment Design—Example

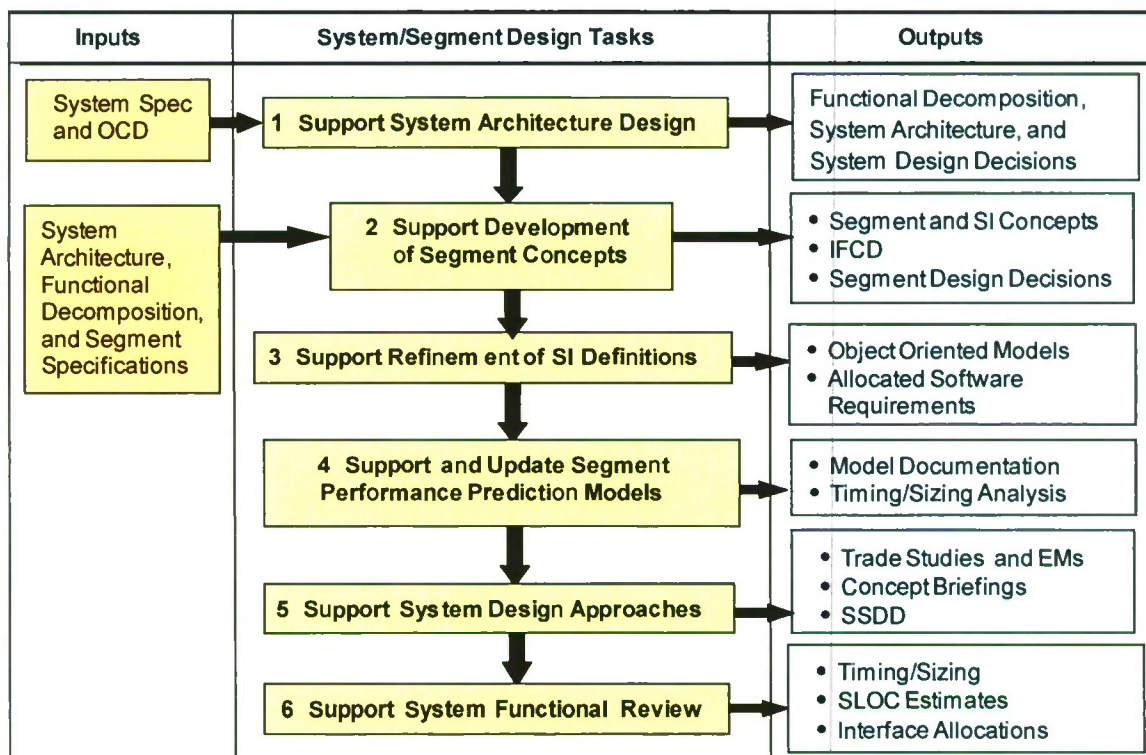
Entry Criteria		Exit Criteria	
Preliminary versions of: <ul style="list-style-type: none"><li>• System and Segment requirements</li><li>• System OCD and Requirements database</li><li>• System test approach and the Verification Cross-Reference Matrix</li></ul>		<ul style="list-style-type: none"><li>• System architecture, Software Item definitions, software system architecture decisions, and non-developmental software analysis are documented.</li><li>• System architecture baseline has been established.</li></ul>	
Verification Criteria			
<ul style="list-style-type: none"><li>• Software personnel participates in the review and approval of the system architecture, SI definitions, and SI interfaces.</li><li>• Program and senior management are provided status of ongoing product engineering tasks (including system design) on a periodic and event driven basis.</li><li>• SQA performs process and product audits for ongoing product engineering tasks per SDP subsection 5.16.</li><li>• System Functional Review (SFR) at the SEIT level successfully completed.</li></ul>			
Measurements			
<ul style="list-style-type: none"><li>• Product Engineering schedule</li><li>• SI SLOC and SI requirements estimates</li></ul>		(see subsection 5.20)	

During the System Design activity, major system characteristics should be refined through trade studies, analyses, simulation, and prototyping. The primary focus of this activity should be the definition of segment Hardware Items (HI) and Software Items (SI). System requirements and interfaces should be refined, allocated, and flowed down to the HI/SI level. In addition, make, buy, and reuse trade studies can be performed during the System Design activity.

The results of these tasks should be used to determine the system characteristics (performance, cost, and schedule) and to provide confidence that risks are being resolved or sufficiently reduced in impact and severity. The System Design activity can also evaluate the maturity of technology and make decisions about the use of technology. This activity is normally led by the SEIT group. The System Test Group **must** review the system/segment design to determine if the requirements allocated are verifiable.

Six principal tasks are recommended for the system design activity as depicted by the example flowchart in Figure 5.4. Details of the six tasks in this activity are described in its related Task Table 5.4-2 that shows the inputs and outputs to each sub-task. In accordance with TOR-3537B, the System/Segment Design activity **must** be described in two paragraphs:

- System-wide/Segment-wide Design Decisions (paragraph 5.4.1)
- System/Segment Architectural Design (paragraph 5.4.2)



OCD = Operational Concept Description

SSDD = System/Subsystem Design Description

SI = Software Items

SLOC = Source Lines of Code

IFCD = Interface Control Document

Figure 5.4. System/Segment Design Process Flow—Example



Table 5.4-2. System/Segment Design Tasks—Example

Tasks	Inputs	Subtasks	Outputs
1. Support System Architecture Design	<ul style="list-style-type: none"> <li>System OCD</li> <li>System Specification</li> </ul>	<ul style="list-style-type: none"> <li>Assist SEIT to identify system level architecture</li> <li>Assist SEIT to develop functional decomposition</li> <li>Coordinate Segment SI definitions with SEIT</li> </ul>	<ul style="list-style-type: none"> <li>System Architecture</li> <li>System Functional Decomposition</li> </ul>
2. Support Development and Update of Segment Concepts	<ul style="list-style-type: none"> <li>System Architecture</li> <li>System Functional Decomposition</li> <li>Segment Specifications</li> </ul>	<ul style="list-style-type: none"> <li>Describe Segment capabilities in context of system specs, HIs, and SIs</li> <li>Describe Segment interfaces to other SIs and elements</li> <li>Describe individual SI capabilities, including plans for reuse of non-developmental software</li> </ul>	<ul style="list-style-type: none"> <li>Segment concepts</li> <li>SI concepts</li> <li>External interfaces captured in the SRS or IRS</li> <li>IFCD</li> </ul>
3. Support Refinement of SI Definitions	<ul style="list-style-type: none"> <li>Segment and SI Concepts</li> </ul>	<ul style="list-style-type: none"> <li>Develop appropriate object-oriented diagrams reflecting all software objects needed to achieve scope with interfaces to other SIs</li> <li>Allocate Segment System requirements to SI classes; verify traceability of system requirements to SIs</li> </ul>	<ul style="list-style-type: none"> <li>OO-Based Models</li> <li>Allocated software requirements</li> </ul>
4. Develop and Update Segment Performance Prediction Models	<ul style="list-style-type: none"> <li>Segment and SI Concepts</li> </ul>	Develop and update Segment models to support: <ul style="list-style-type: none"> <li>Updated timing and sizing analysis</li> <li>Algorithm development</li> <li>Interface analysis</li> </ul>	<ul style="list-style-type: none"> <li>EMs documenting timing and sizing analysis</li> <li>Performance prediction models and documentation</li> </ul>
5. Develop Design Approaches	<ul style="list-style-type: none"> <li>Segment Specs and SI Concepts</li> <li>Performance Prediction Models</li> <li>Segment OO Models</li> </ul>	<ul style="list-style-type: none"> <li>Create Segment-level behavior diagrams for key design approaches</li> <li>Verify approach satisfies associated system requirements</li> <li>Support documentation of technical approaches and the System/Segment Design Description (SSDD)</li> </ul>	<ul style="list-style-type: none"> <li>Trade Study EMs</li> <li>Concept briefings</li> <li>SSDD</li> </ul>
6. Perform System Functional Review (SFR)	<ul style="list-style-type: none"> <li>Concept Briefings</li> <li>Segment OO Models</li> </ul>	Conduct analysis to: <ul style="list-style-type: none"> <li>Allocate timing and sizing budgets to SIs</li> <li>Establish and update SLOC estimates</li> <li>Allocate system and external interfaces</li> <li>Flow up changes to system requirements as needed</li> </ul>	<ul style="list-style-type: none"> <li>Timing and sizing budgets</li> <li>Updated SLOC estimates</li> <li>Interface allocations</li> </ul>

### 5.4.1 System-wide/Segment-wide Design Decisions

**System Decisions.** System-wide software design decisions and their rationale should be documented by the SEIT in Engineering Memorandums (EMs) and the System/Subsystem Design Description (SSDD). EMs are typically maintained in the electronic data management system (see paragraph 5.2.3). System requirements are generated from the EMs and the SSDD and are flowed down to the segment product specifications.

**Segment Decisions.** Segment-wide software design decisions and their rationale should be documented in EMs residing in the electronic data management system. Segment EMs **must** be evaluated by the segment IPT to determine if they impact the software requirements. However, the principal product is the SSDD. The segment Software IPT should record, in the segment SDF or equivalent, the rationale for the COTS/reuse/other NDIs approach selected—including rejected approaches and the studies and analyses that led to the selected approach.

Software system/segment architecture decisions made during system/segment design **must** be recorded for later use in developing software requirements and design. Decisions are usually recorded using EMs. The Segment IPT Lead should participate in establishing the rationale for software



architecture, definitions, interfaces, COTS/reuse/other NDIs approach, and should be responsible for ensuring that the following data are recorded:

- The overall software architecture that was selected, including the studies and analyses that lead to the selected architecture
- The Software Item definitions and interfaces, including the studies and analyses that lead to the selected SI definitions
- The software COTS/reuse/other NDIs approach, including studies and analyses that led to the selected approach

#### **5.4.2 System/Segment Architectural Design**

This task involves organizing a system into segments/subsystems and then decomposing segments or subsystems into Hardware Items (HIs), Software Items (SIs), plus manual and other operations.

##### **5.4.2.1 System Architectural Design**

During the System Architectural Design task, required segments **must** be identified along with segment-to-segment, and segment-to-external systems interfaces, plus a concept of system execution. The interfaces **must** be documented by the SEIT in the IFCDs. The segment software IPT personnel should support the SEIT by:

- Participating in the system architectural design and the specific configuration of hardware, software, and firmware to meet performance and reliability requirements
- Assessing the software impact of implementing the operational concept and system requirements in terms of technical suitability, cost, and risk
- Participating in trade studies to select processing, communications, and storage resources
- Reviewing the system test approach and test philosophy to ensure testing compatibility
- Identifying how each requirement will be tested, what test support software will be needed for system test, identifying the system test environment, and developing a system test plan
- Reviewing and analyzing the system design to determine testability of the requirements allocated to software
- Recommending requirements changes to the SEIT as necessary

##### **5.4.2.2 Segment Architectural Design**

Segment architectural design should be documented as part of the segment architecture baseline process. The basic responsibilities, typically assigned to segment software IPT personnel, during Segment Architectural Design, are to:

- Support System Engineering in defining the segment architectural design and the specific configuration of hardware, software, and firmware to meet performance and reliability requirements
- Assess the software impact of implementing the operational concept and segment requirements for technical suitability, cost, and risk
- Support System Engineering in performing trade studies to select best processing, communications, and storage resources
- Ensure the segment test approach is compatible with the test philosophy
- Support segment system engineering in reviewing the segment-level requirements and identifying how each requirement is tested

- Support the identification of test support software and test environment needed for segment-level test and development of a segment test plan
- Review and analyze segment design to determine testability of requirements allocated to software
- Recommend requirements changes to segment system engineering
- Support segment system engineering in creating definitions of Software Items, in allocating segment requirements to the SIs, and in review and refinement of the interfaces among the defined software products
- Identify potential candidates for reuse and COTS software products at the SI level
- Review and refine the definition of Software Items

When using the object-oriented (OO) methodology, the segment software high level architecture design **must** be captured in SI OO models and placed in the SDF. These architecture models should be documented with the applicable OO methodology products. These products should then be utilized to refine and update the development of timing and sizing budgets, SLOC estimates, and to prototype algorithmic approaches.

## 5.5 Software Requirements Analysis

**Introduction.** This subsection of the SDP addresses the objectives, approach, work products, and responsibilities of the Software Requirements Analysis activity. There are no specific TOR-3537B SDP sectional breakdowns required for this activity.

**Objectives.** Software requirements analysis **must** be accomplished by analyzing the system and segment requirements to identify allocated and derived software requirements. These resulting requirements should define *what* the software system must be able to do, while avoiding implementation bias (i.e., not describe *how* to implement a requirement).

As depicted in Figure 5.5-1, software requirements can originate from several sources. The types of requirements that may be determined include, but are not limited to, capabilities, behavior, processing, control, interfaces, performance, sizing, timing, packaging, security, safety, reliability, maintainability, availability, human factors, and software qualification. Requirements should be analyzed for: completeness, traceability, consistency, testability, criticality, feasibility, correctness, and accuracy.

In addition, requirements **must** be specifically evaluated for safety, security, privacy protection, dependability, reliability, maintainability, and availability. These critical requirements need additional tracking and monitoring per paragraph 4.2.5. In addition, requirements identified that have significant risk associated with them **must** be evaluated by the software IPT for risk assessment and mitigation in accordance with SDP subsection 5.19.

**Approach.** The general approach for preparing requirements specifications is to make each requirement:

- Clear and concise in a single statement with a single “shall”
- Testable or verifiable and traceable via a unique product identifier
- Consistent with all system requirements
- Understandable and independent of other software requirements
- A statement of what the software will do and not how it should do it

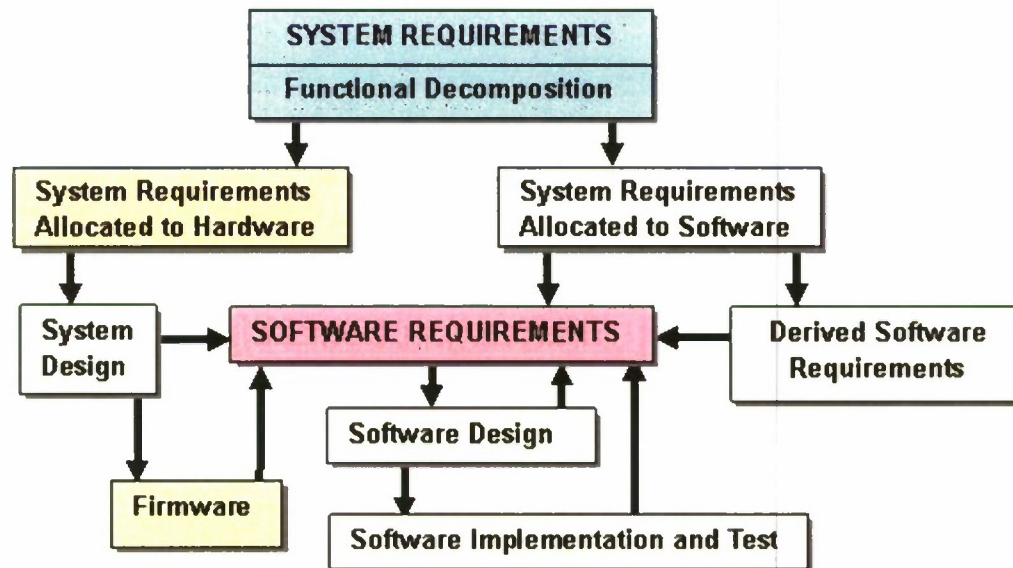


Figure 5.5-1. The Origin of Software Requirements

All software requirements may not meet all of these guidelines, however, these guidelines should be considered in defining each requirement. In addition to being clear and concise in documenting the requirements, analysts should: use data dictionary terms and approved acronyms; use consistent terminology; avoid the use of lists; limit the use of the words “and” and “or;” use positive requirements; and not use the term “and/or.”

A single requirements database should be used to capture all requirements. System Engineering and Software Engineering should use the same requirements database for documenting and maintaining requirements to assure full compatibility between these tasks at the system, segment and software levels. Portions of this database can be partitioned out and updated by the various IPTs; however, a single master copy should be maintained to ensure a consistent communication of requirements among the teammates.

Table 5.5-1 summarizes the readiness criteria for this activity in terms of the entry and exit criteria, verification criteria to ensure completion, and the measurements usually collected.

**Software Work Products.** One Software Requirements Specification (SRS) should be developed for each segment SI. The segment Software IPT personnel **must** ensure that there is consistency between software item SRSs for common interface requirements. SS-2 software items do not require an SRS document; however, all software and interface requirements and traceability data for SS-2 software **must** be captured in the SDF.

The software segments should document allocated and derived software requirements in a Requirements Database, as discussed in SDP paragraph 4.2.3, for all software categories. Each specific requirement **must** be assigned a unique program identifier for individual requirement traceability. Traceability requirements for MC-1 and SS-1 software should include SRS requirements traced to system requirements and software builds.



Table 5.5-1. Readiness Criteria: Software Requirements Analysis—Example

Entry Criteria	Exit Criteria
<ul style="list-style-type: none"> <li>System requirements allocated to software are available.</li> <li>Software system architecture is available.</li> <li>System OCD is available.</li> <li>Appropriate Software Engineering Environment (SEE) elements are available for use.</li> </ul>	<ul style="list-style-type: none"> <li>Software system architecture and software system interfaces are documented in the SDF, SRS, and IRS.</li> <li>Software requirements, Requirements Test Verification Matrix (RTVM), and the Software Requirements Traceability Matrix (SRTM) are documented in, or their location referenced by the SRS, or in a traceability or requirements management tool.</li> <li>Lessons learned are recorded.</li> </ul>
Verification Criteria	
<ul style="list-style-type: none"> <li>Software management reviews and approves: software system architecture, software system interfaces, software requirements, RTVM and SRTM as documented in the SRS, or their location referenced by the SRS, or in a traceability or requirements management tool.</li> <li>Program and senior management are provided status of ongoing product engineering tasks (including software requirements) on a periodic and event driven basis.</li> <li>All software products (see Table 5.5-2) are peer reviewed and the requirements database is inspected.</li> <li>A Software Specification Review (SSR) or Technical Interchange Meeting (TIM) has been completed.</li> </ul>	
Measurements	
<ul style="list-style-type: none"> <li>Software requirements added, modified, and deleted during reporting period</li> <li>Product Engineering schedule</li> <li>Requirements traceable versus untraceable.</li> </ul>	
(see subsection 5.20)	

The traceability data **must** be documented and the recommended format is a Software Requirements Traceability Matrix (SRTM). In addition, a Requirements Test Verification Matrix (RTVM) should also be prepared. The RTVM may also be called the Verification Cross Reference Matrix (VCRM). The SRTM and the RTVM (or VCRM) may part of the SRS or reside in a traceability or requirements management tool. **It is extremely important to include government overview in this process** to assure simulators are developed with adequate requirements from the stakeholders and users to incorporate the needed fidelity.

Diagrams for algorithm models and simulations should be captured in SDFs as well as the appropriate software tools. For COTS/Reuse (C/R) software, the only documentation available may be from the software supplier. However, MC and SS-1 categories of COTS/Reuse software (defined in paragraph 1.2.3) once integrated **must** be fully documented in order to pass the software documentation reviews.

**All of these software products must be made available via an Electronic Data Interchange Network (EDIN)** as described in SDP subparagraph 5.2.3.1. In addition, the draft of the Software Master Build Plan (SMBP) may be prepared during this activity and the Interface Control Document (IFCD) may be baselined if it had not reached that level of maturity in the previous activity. The typical software products for the Software Requirements analysis activity are summarized in example Table 5.5-2.

Table 5.5-2. Software Requirements Analysis Work Products—Example

Software Requirements Analysis Products	MC1	SS1	SS2	SS3	C/R
Requirements Database	Required	Required	Required	Required	Required
Software Requirements Specification (SRS) (May include IRS, SRTM & RTVM)	Required	Required	Required*	Required*	Required*
Interface Requirements Specification (IRS)	Required	Required	Required*	Required*	Required*
Software Requirements Traceability Matrix (SRTM)	Required	Required	Required*	Required*	Required*
Requirements Test Verification Matrix (RTVM)	Required	Required	Required*	Required*	Required*
Software Master Build Plan (SMBP)	Required	Required	Required*	Required*	Required*
Software Work Products (see subparagraph 4.2.10.3)	Required	Required	Required*	Optional	Optional

\*Document not required, but applicable information is developed and retained in the SDF.

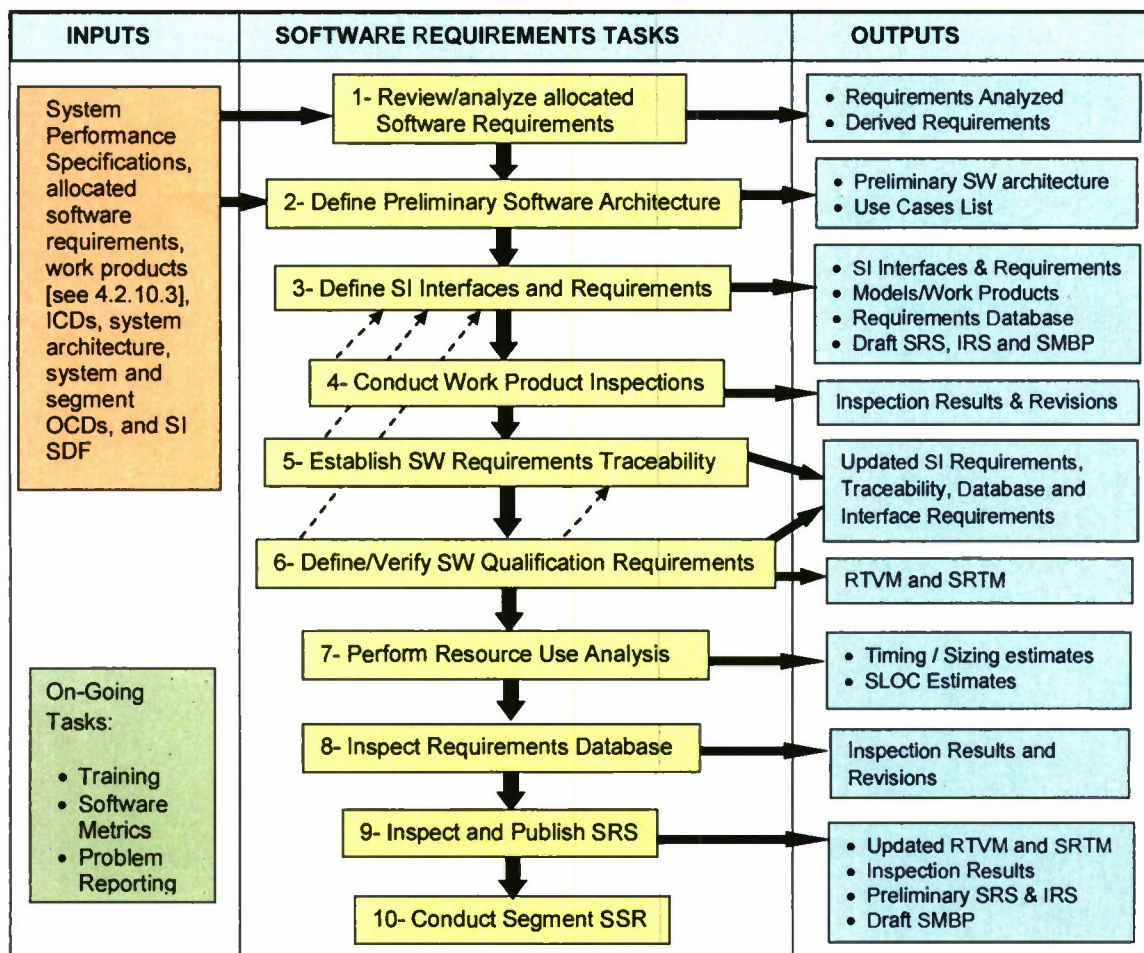
**Staff Responsibilities.** The segment Software IPT personnel should be responsible for the SI requirements analyses, the generation of the work products, and the documentation of the requirements in SRSs or SDFs, as appropriate. Table 5.5-3 is an example of the roles and responsibilities of the Software IPT personnel and other groups during the requirements analysis activity. The segment Software IPT personnel should also be responsible for conducting the required reviews of the analysis process and output documentation as well as the tasks defined in the example flowchart (Figure 5.5-2) and in its related Task Table (Table 5.5-4).

Table 5.5-3. Roles and Responsibilities During Software Requirements Analysis—Example

Roles	Responsibilities
<b>Software IPT</b>	Performs software requirements analysis, definition, and documentation
	Generates initial traceability products for SRS requirements to parent specification requirements in Requirements Database
	Identifies software item risk areas
	Initiates an SDF for each SI
	Collects and reports requirements metrics
	Submits problem reports after the requirements documentation is baselined
<b>Software Test</b>	Identifies the verification levels and methods in the Requirements Database; prepares RTVM
<b>Chief Software Engineer</b>	Supports the IPT in the identification and specification of critical requirements, reviews the SDFs, reviews the activity products, and attends all formal activity reviews
<b>Software Quality Assurance (SQA)</b>	Evaluates the segments for adherence to: (a) documented policies and procedures; (b) product quality criteria; and (c) the Requirements Database. Findings are reported to management.
<b>Software Configuration Management (SCM)</b>	Manages software requirements baseline and processes all Software Discrepancy Reports/Software Change Notices (SDRs/SCRs) for documented software requirements changes to the SRS or SDF as they are generated by the software segments or elements.
<b>IPT CCB</b>	Addresses all segment internal change notices and SDRs/SCRs as they are generated



**Process Tasks.** The software requirements analysis activity involves 10 tasks as depicted in Figure 5.5-2. Details of these 10 tasks are described in Table 5.5-4 in terms of the inputs and outputs to each task.



NOTE: Updated SDFs are outputs for each activity

OCD = Operational Concepts Description  
 ICD = Interface Control Document  
 RTVM = Requirements Test Verification Matrix  
 SRS = Software Requirements Specification

SDF = Software Development Folders  
 IRS = Interface Requirements Specification  
 SRTM = Software Requirements Traceability Matrix

SLOC = Software Lines of Code  
 SMBP = Software Master Build Plan

Figure 5.5-2. Software Requirements Analysis Process Flow—Example



Table 5.5-4. Software Requirements Analysis Tasks—Example

Task	Inputs	Subtasks	Outputs
1. Review and Analyze Allocated Software Requirements	<ul style="list-style-type: none"> <li>See inputs in Figure 5.5-2</li> </ul>	Review and analyze allocated SW requirements	<ul style="list-style-type: none"> <li>Allocated requirements analyzed</li> <li>Derived requirements developed</li> </ul>
2. Define Preliminary Software Architecture	<ul style="list-style-type: none"> <li>Allocated Software Requirements</li> </ul>	<ul style="list-style-type: none"> <li>Define software architecture components</li> <li>Develop/update SI-to-SI interfaces</li> <li>Identify segment Use Cases</li> </ul>	<ul style="list-style-type: none"> <li>Preliminary SW architecture at SI level</li> <li>Segment Use Cases List</li> </ul>
3. Define SI Interfaces and Requirements	<ul style="list-style-type: none"> <li>ICDs</li> <li>Software Requirements</li> <li>Software Architecture at SI Level</li> </ul>	<ul style="list-style-type: none"> <li>Refine SI level SW architecture model</li> <li>Model software architecture in OO</li> <li>Develop SW requirements and interface requirements, including data items</li> <li>Develop SW work products (see subparagraph 4.2.10.3 and Work Product Table 5.5-2)</li> <li>Identify software risks</li> <li>Enter SW and interface requirements, into database</li> <li>Review database for completeness</li> </ul>	<ul style="list-style-type: none"> <li>High level analysis and class models</li> <li>SI and interface requirements</li> <li>Work products</li> <li>Draft SRS, IRS, and SMBP</li> <li>Populated Requirements Database</li> </ul>
4. Conduct Work Product Inspections	<ul style="list-style-type: none"> <li>Work Products</li> <li>Requirements Database</li> </ul>	<ul style="list-style-type: none"> <li>Schedule inspection; distribute review package</li> <li>Conduct peer reviews; verify feasibility, completeness of SW requirements, and consistency between SW requirements, and update work products</li> <li>Document results and post to SDF</li> <li>Fix inspection deficiencies</li> </ul>	<ul style="list-style-type: none"> <li>Peer Review results</li> <li>Work products updated</li> <li>Deficiencies recorded</li> </ul>
5. Establish Software Requirements Traceability	<ul style="list-style-type: none"> <li>Requirements Database</li> </ul>	<ul style="list-style-type: none"> <li>Update software requirements tables to add traceability between system and software requirements</li> <li>Verify both downwards and upwards traceability between system and software requirements</li> <li>Add software requirements or flow up recommended changes to system requirements as necessary to complete traceability</li> <li>Conduct peer review of Requirements Database</li> </ul>	<ul style="list-style-type: none"> <li>Updated SI requirements</li> <li>Updated traceability</li> <li>Updated Requirements Database</li> <li>Updated interface requirements</li> </ul>
6. Define/Verify Software Qualification Requirements	<ul style="list-style-type: none"> <li>Draft Software Requirements</li> </ul>	<ul style="list-style-type: none"> <li>Update software requirements tables to add a qualification method (inspection; analysis; test; demonstration; other) for each software requirement</li> <li>Verify qualification method satisfies verification plan</li> <li>Create and peer review the RTVM and SRTM</li> </ul>	<ul style="list-style-type: none"> <li>RTVM and SRTM</li> <li>Updated SI requirements, traceability, interfaces, and database</li> </ul>
7. Perform Resource Use Analysis	<ul style="list-style-type: none"> <li>Software Work Products, Requirements, and Preliminary Architecture</li> </ul>	<ul style="list-style-type: none"> <li>Conduct timing and sizing analysis</li> <li>Develop/update SLOC estimates</li> <li>Post information to SDF</li> <li>Verify that timing and sizing meets requirements</li> </ul>	<ul style="list-style-type: none"> <li>Timing and sizing estimates</li> <li>SLOC estimates</li> </ul>
8. Inspect Requirements Database	<ul style="list-style-type: none"> <li>Requirements Database</li> </ul>	<ul style="list-style-type: none"> <li>Announce inspections, disseminate schedule, and review products in advance</li> <li>Conduct inspection to verify correctness, completeness, and consistency of data</li> <li>Document results and post to SDF</li> <li>Fix inspection deficiencies</li> </ul>	<ul style="list-style-type: none"> <li>Requirements Inspection results</li> <li>Baselined Requirements Database</li> </ul>
9. Inspect and Publish SRSs	<ul style="list-style-type: none"> <li>Requirements Database</li> </ul>	<ul style="list-style-type: none"> <li>Update the preliminary SRS and IRS</li> <li>Conduct peer reviews utilizing SRS inspection criteria</li> <li>Update the draft SMBP</li> <li>Obtain board approval</li> </ul>	<ul style="list-style-type: none"> <li>Preliminary SRSs and IRSs</li> <li>Updated SI SDF</li> <li>Updated RTVM and SRTM</li> <li>Draft SMBP</li> </ul>
10. Conduct Segment SSR	<ul style="list-style-type: none"> <li>SRSs/IRSs</li> <li>Agenda</li> <li>Presentation slides</li> </ul>	<ul style="list-style-type: none"> <li>Conduct SSR</li> <li>Publish minutes and action items list</li> <li>Resolve action items</li> <li>Deliver documentation per contract</li> </ul>	<ul style="list-style-type: none"> <li>SSR minutes</li> <li>Action item results</li> <li>SRSs and IRSs</li> <li>Requirements Database</li> </ul>

**Exit Criteria.** There is no requirement for the software requirements analysis activity to be entirely completed prior to the start of the software design activity. When following iterative lifecycle models, the software requirements analysis activity may be repeated for each build so the software requirements would be developed iteratively. In this case, the Software Specification Review (SSR), or the support software requirements Technical Interchange Meeting (TIM), would be held on a build-by-build basis.

When following the Object-Oriented Analysis (OOA) methodology, the software requirements analysis and software architecture definition can be concurrent resulting in combining the SSR, or TIM, with the Preliminary Design Review (PDR).

The software requirements analysis activity formally ends upon completion of the Software Specification Review (SSR) and baselining of all work products. Lessons learned should be captured in EMs and SDFs.

**Verification.** Verifying completion of the 10 tasks described in this activity is accomplished by a combination of approvals by software leads, peer reviews, SQA audits, periodic audits by the CSWE and Joint Technical Reviews (JTR) or TIMs as determined to be necessary for each task.

## 5.6 Software Design

**Introduction.** This subsection of the SDP addresses the objectives, approach, documentation, and staff responsibilities of the Software Design activity. In accordance with TOR-3537B, the Software Design activity **must** be described by three paragraphs in the SDP:

- **Software Item-wide Design Decisions:** The software designers define the Software Item (SI)-wide design decisions that affect the selection and design of the Software Units (SU) comprising the SI. (paragraph 5.6.1)
- **Software Item Architectural Design:** Software designers develop an architectural design that partitions the SI into SUs or SUs that may be subdivided into smaller SUs. (paragraph 5.6.2)
- **Software Item Detailed Design:** Software designers perform a detailed design on the individual SUs and produce a description of the SI down to the level of algorithms and procedures. (paragraph 5.6.3)

The process specified in the three tasks pertain to MC-1 and SS-1 software only. For the SS-2 software category, the three tasks are often combined into a single activity.

**Objectives.** Software Architectural Design is the first focused design activity at the SI level. Architectural decisions applicable to all SIs should have been made during the System Design activity (see subsection 5.4). Architecture at the SI level **must** determine the design for interfacing to other SIs and to hardware units (if appropriate). Human interfaces may be designed (e.g., using prototypes to validate the designs with end users). The tasking or operating system process structure for the SI should be determined in this activity. Additional site-specific products and design reviews may be specified in the segment's site-specific SDP Annexes.



**Approach.** The three major tasks of the design activity, as defined by TOR-3537B, are intended to be performed as consecutive steps of increasing levels of design specificity. However, **there is no TOR-3537B requirement for each activity to be completed for the entire SI before the next design activity is started.** These three tasks usually overlap each other, and when using the iterative lifecycle model, the three tasks are usually performed iteratively for each build. Table 5.6-1 summarizes the readiness criteria in terms of entry and exit criteria, verification criteria to ensure completion of the required tasks, and the measurements usually collected during the design activity.

Table 5.6-1 Readiness Criteria: Software Design—Example

Entry Criteria	Exit Criteria
<ul style="list-style-type: none"> <li>• Software requirements are allocated to the SI and approved.</li> <li>• Software system architecture has been approved.</li> <li>• System architecture and the OCD are available.</li> <li>• System verification matrix is available in the Requirements Database.</li> <li>• Software use cases and scenarios, SI definitions, interface design, updated Requirements Database, and preliminary database architecture are documented in the SDF.</li> </ul>	<ul style="list-style-type: none"> <li>• Software architecture and design are captured in design models.</li> <li>• Performance and sizing analyses are documented in engineering memos.</li> <li>• SI SLOC estimates are updated.</li> <li>• For MC-1 and SS-1 software, a baselined STP, SAD, SDD, SMBP, DBDD, and IDD are ready.</li> <li>• Design is baselined and placed under SCM control.</li> </ul>
Verification Criteria	
<ul style="list-style-type: none"> <li>• Program management is provided status of ongoing product engineering tasks on a periodic and event driven basis.</li> <li>• SQA performs process/product audits for ongoing product engineering tasks per SDP subsection 5.16.</li> <li>• All software architecture and design work products are peer reviewed and measurements documented.</li> <li>• A preliminary technical review of the architecture and design has been completed and the software PDR and CDR (or TMS) are completed.</li> </ul>	
Measurements	
<ul style="list-style-type: none"> <li>• Product Engineering schedule (including software architecture and design tasks)</li> <li>• Results from peer reviews</li> <li>• SLOC estimates</li> </ul>	
(see subsection 5.20)	

**Software Work Products.** The documentation produced during the Software Design activity for each SI includes the: software architecture and design and interface design descriptions; test plan, models and diagrams; traceability products in the Requirements Database; and the Software Master Build Plan (SMBP) that maps each build to the capabilities provided by the build and specific requirements allocated to the build.

Software interface design descriptions **must** be documented in the Software Design Description (SDD), Software Architecture Description (SAD), and the Interface Design Document (IDD). The SDD documents the SI design decisions and the SAD documents the SI architectural design, and design of each SU. (Note that section 4 of the SDD describes the architectural design; the SAD replaces that section by including much more information, such as multiple architectural views). All software and interface design work products **must** be recorded in the SDF. Design documentation for C/R software is limited to data provided by the vendor.

Diagrams for algorithm models and simulations, initiated during the software requirements definition activity, should be expanded and refined during the Software Design activity. The revised work models and diagrams **must** be maintained in the appropriate software tools and captured in the SDF. The SRS, baselined in the previous activity, may be updated. Also, during the detailed design portion of this activity, draft versions of the Software Users Manual (SUM as described in subparagraph 5.12.3.1) and the Software Transition Plan (STrP as described in paragraph 5.13.9) may also be prepared.



MC-1 and SS-1 software require the traceability of the software architecture and design elements from the SAD, SDD, and DBDD to the software requirement unique project identifiers in the SRSs. Also, IDD elements are required to be traced to SRS or IRS requirements. This traceability information **must** be documented in a Requirements Database as discussed in SDP paragraph 4.2.3.

An example of required work products for the Software Design activity is summarized in Table 5.6-2. These software products **must** be made accessible through an Electronic Data Interchange Network (EDIN) as described in SDP subparagraph 5.2.3.1.

Table 5.6-2. Required Software Design Activity Work Products—Example

Software Design Products	MC-1	SS-1	SS-2
SAD and SDD (per Software Item per build)	Required	Required	Required*
IDD and SMBP (per build)	Required	Required	Required*
STP (per Software Item)	Required	Required	Required*
DBDD (if required)	Required	Required	Required*
SDF capturing revised models and diagrams	Required	Required	Required
Software design elements traced to SRS requirements in the Requirements Database	Required	Required	Optional

\*Document not required, but applicable information is developed and retained in the SDF.

**Roles and Responsibilities.** The segment IPT software personnel **must** be responsible for the Software Item architectural and detailed designs, the revision of design work products, and the documentation of the design in SADs, SDDs, IDDs, DBDDs, and SDFs, as appropriate. Table 5.6-3 is an example of the roles and responsibilities for Software IPT personnel and other groups during the software design activity.

Table 5.6-3. Roles and Responsibilities During Software Design—Example

Roles	Responsibilities
Software IPT	Conducts required reviews of the software architecture and design process and develops outputs
	Updates and maintains the SDF
	Addresses critical software requirements in the software architecture and design
	Generates traceability products for design elements to SRS requirements unique project identifiers in the Requirements Database
	Identifies software architecture and design risk areas and provides identified risks to management
	Collects and reports software architecture and design activity metrics
	Generates computer hardware resource utilization estimates, comparing to the required threshold values, and addresses estimates that exceed the requirements
	Submits SCRs or SDRs, as necessary, after design documentation is baselined
Software Test	Initiates the STP
CSWE	Supports the segment IPT software personnel in the handling of security and critical requirements in the software design, audits SDFs, reviews activity products, attends all formal activity reviews, and monitors and analyzes software metrics
SQA	Evaluates the segment IPT software personnel for adherence to documented policies and procedures, evaluates segment IPT software architecture and design work products for product quality, and documents and report findings to upper-level management
SCM	Processes all SCRs and SDRs for software architecture and design changes to the baselined SAD, SDD, IDD, DBDD, SMBP, and STP documented design as they are generated

### 5.6.1 Software Item-wide Design Decisions

**Objectives.** The objective of this first task in the Software Design activity is to define and record the Software Item-wide design decisions. These decisions constrain how the designers partition the SIs into SUs and overall design of the SUs. These are global decisions about the structure of the design that impact the SIs.

**Approach.** The Software Design activity normally begins by performing an examination of the requirements relative to the SI plans, environment and interfaces to determine if there are any SI-wide design issues. Where such issues are identified, segment IPT software personnel should analyze the issues and determine an appropriate design constraint or decision for each. These design decisions **must** then be documented and communicated to the software designers as a set of design constraints in conjunction with the requirements of what they are to design.

Design decisions are program specific, however, key factors that may be considered in determining SI design issues include:

- Safety, security, and privacy-critical requirements
- Computer hardware platform and resource utilization requirements
- External SI constraints and interfaces
- Algorithms and Application Program Interfaces (API) to be used
- Uniform exception handling and recovery methods
- Major architectural trade-offs
- Applicable standards and Graphical User Interfaces (GUI)
- Proposed software product reuse
- Uniform data storage and access methods
- Performance characteristics including response times, software maintainability, reliability, and availability not allocated to individual architecture components
- Human factors, training requirements, and SI operational constraints

Key design decisions identified, and the rationale for making those decisions, **must** be documented in the SAD, SDD, DBDD, and IDD for MC-1 and SS-1 software and in the SDFs for SS-2. Key design decisions are those that could impact or constrain the SI Architectural Design, SI to external interfaces, software requirements, cost or schedule. Design decisions for SS-2 software should be reviewed during design inspections. For multiple build SIs, design decisions should be addressed prior to completion of the Detailed Design for the first build. Design Decision tasks are integrated into the flowcharts and task tables as described in SDP paragraphs 5.6.2 and 5.6.3.

## 5.6.2 Software Item Architectural Design

**Objectives.** The objective of SI Architectural Design is to describe the high-level organization of the SIs in terms of SUs and their relationships. The IPT developing the SI **must** prepare an architecture that meets the system requirements. The main objectives of SI Architectural Design are to:

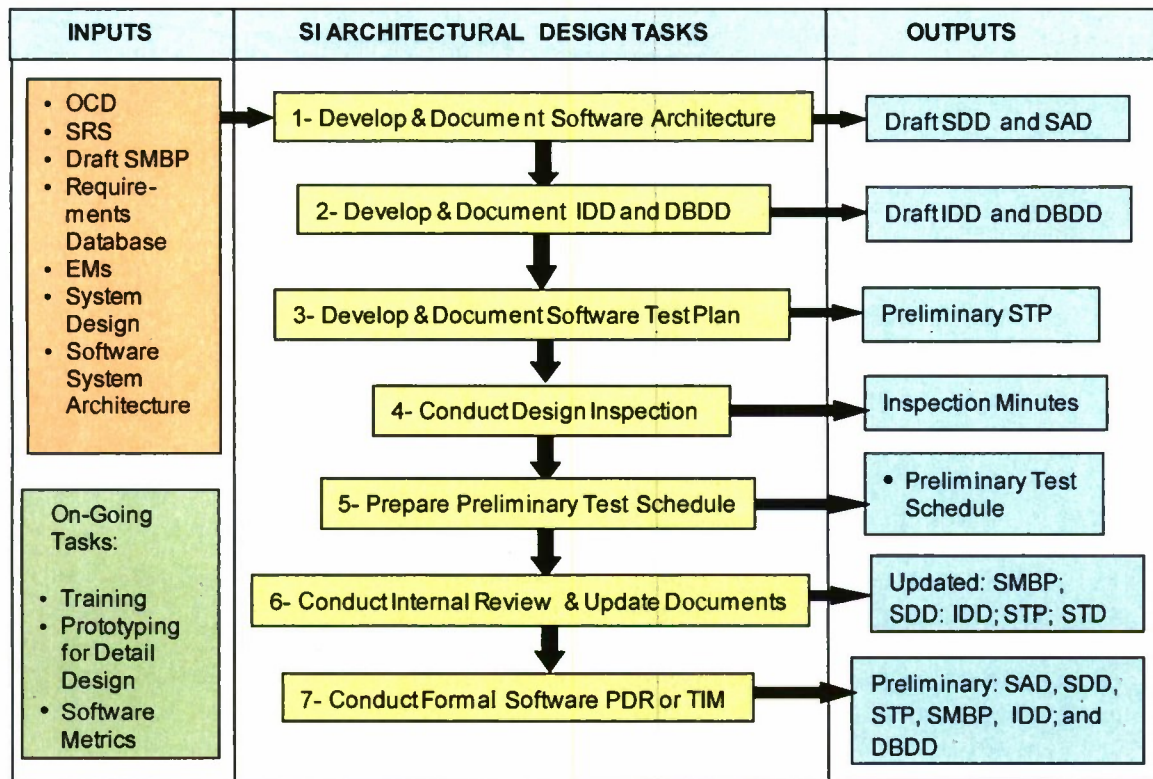
- Decompose the SIs design into SUs
- Allocate requirements from SRSs to SUs
- Complete allocation of requirements from the SRS to Use Cases (for OOD)
- Describe the architectural design and requirements allocation in a preliminary SAD and SDD
- Update the SI SDF for the SI and update the baselined SRS if necessary
- Prepare the applicable preliminary STP, SAD, SDD, IDD, SMBP, DBDD and update the SRS and IFCD as needed

**Approach.** Software Item Architectural Design **must** be performed by segment IPT software personnel. Using the documented software requirements, and the initial work products (models and diagrams) from the requirements definition activity, the software architecture models are refined and



the architectural components, including Software Units (SUs), are identified. SUs are logical constructs for classes and associations in OOA/OOD or specific capabilities in a structured development. Use of graphical architecture modeling techniques, e.g., Unified Modeling Language (UML), is required.

The principal tasks, recommended for the SI Architectural Design Process, are depicted in Figure 5.6.2 in flowchart form and in its related Task Table 5.6.2 in terms of the inputs and the outputs of each task.



OCD = Operational Concepts Description  
 SRS = Software Requirements Specification  
 SMBP = Software Master Build Plan  
 EMs = Engineering Memos  
 SDD = Software Design Description  
 SAD = Software Architecture Description

IDD = Interface Design Document  
 STP = Software Test Plan  
 DBDD = Data Base Design Description  
 PDR = Preliminary Design Review  
 TIM = Technical Interface Meeting

Figure 5.6.2. Software Item Architectural Design Process Flow—Example



Table 5.6.2. Software Item Architectural Design Tasks—Example

Task	Inputs	Subtasks	Outputs
1. Develop and Document Software Architecture	<ul style="list-style-type: none"> <li>SRS</li> <li>Requirements Database</li> <li> OCD</li> <li> EMs</li> <li> System Design</li> <li> Draft SMBP</li> <li> Software System Architecture</li> </ul>	<ul style="list-style-type: none"> <li>Determine SI modes of operation and architectural approach</li> <li>Perform analysis of reusable software and allocate to SIs</li> <li>Define software functions, behavior, error conditions, services, and controls</li> <li>Identify architectural components including SUs</li> <li>Prepare applicable OO or SA/SD models</li> <li>Perform resource use analysis of timing and sizing budgets</li> <li>Allocate requirements from SRS to SUs and Use Cases</li> <li>Allocate SUs to processors and determine protocols</li> <li>Update RTVM with links to design components</li> <li>Prepare draft SDD and SAD</li> <li>Conduct internal review of software architecture</li> </ul>	<ul style="list-style-type: none"> <li>Draft SDD and SAD</li> <li>OO Models</li> <li>SA/SD Models</li> </ul>
2. Develop and Document Interface Design	<ul style="list-style-type: none"> <li>Preliminary SDD</li> </ul>	<ul style="list-style-type: none"> <li>Allocate requirements to SUs and Use Cases</li> <li>Define software internal interfaces</li> <li>Update software external interfaces and RTVM</li> <li>Prepare draft IDD</li> <li>Define database logical design and the draft DBDD</li> <li>Conduct internal review of software interface design</li> </ul>	<ul style="list-style-type: none"> <li>Draft IDD</li> <li>OO Models</li> <li>SA/SD Models</li> <li>Draft DBDD</li> </ul>
3. Develop and Document STP	<ul style="list-style-type: none"> <li>Draft SDD, SAD, and IDD</li> </ul>	Prepare preliminary Software Test Plan (STP) based on System/Segment Test Plan	<ul style="list-style-type: none"> <li>Preliminary STP</li> </ul>
4. Conduct Design Inspection	<ul style="list-style-type: none"> <li>SAD, SDD, STP, and IDD</li> <li>Requirements Database</li> <li>Design Models</li> </ul>	As defined in SDP subsection 5.15: <ul style="list-style-type: none"> <li>Inspect links to the design in the Requirements Database</li> <li>Inspect design work products</li> <li>Perform document reviews of the SDD and the IDD</li> </ul>	<ul style="list-style-type: none"> <li>Inspection Minutes</li> </ul>
5. Prepare Test Schedule	<ul style="list-style-type: none"> <li>Preliminary STP SAD, SDD, and IDD</li> </ul>	<ul style="list-style-type: none"> <li>Identify threads and prepare the preliminary schedule for integrating threads on target hardware</li> </ul>	<ul style="list-style-type: none"> <li>Preliminary Test Schedule</li> </ul>
6. Conduct Internal Review and Update Documents	<ul style="list-style-type: none"> <li>Preliminary SAD, SDD, IDD, and STP</li> <li>Draft DBDD</li> </ul>	<ul style="list-style-type: none"> <li>For MC-1 software, segment SI IPT conducts an internal segment software PDR with management</li> <li>For SS-1 &amp; SS-2 software, segment SI IPT conducts an internal segment software TIM with management</li> <li>Update the SAD, SDD, IDD, DBDD, and STP as required</li> </ul>	<ul style="list-style-type: none"> <li>Updated SDD, IDD, STP, and SMBP</li> <li>Incorporate SS-2 data into SDF</li> </ul>
7. Conduct Software PDR (MC-1) or TIM (SS-1)	<ul style="list-style-type: none"> <li>Updated SDD, IDD, STP, and SMBP</li> </ul>	<ul style="list-style-type: none"> <li>Schedule the PDR/TIM, identify attendees, and finalize agenda</li> <li>Conduct the PDR/TIM and generate minutes and action items</li> <li>Ensure closure of action items and generate final outputs</li> </ul>	<ul style="list-style-type: none"> <li>Preliminary SAD, SDD, IDD, STP, SMBP, and DBDD</li> </ul>

The Software Test Plan (STP) is usually produced concurrently with the Software Item Architectural Activity (SDP paragraph 5.6.2) and is shown as Task 3 in both Figure 5.6.2 and Table 5.6.2.

Production of the STP is actually a product of the Software Item Test Planning activity (see paragraph 5.1.2) and it is prepared by the software test engineers.

### 5.6.3 Software Item Detailed Design

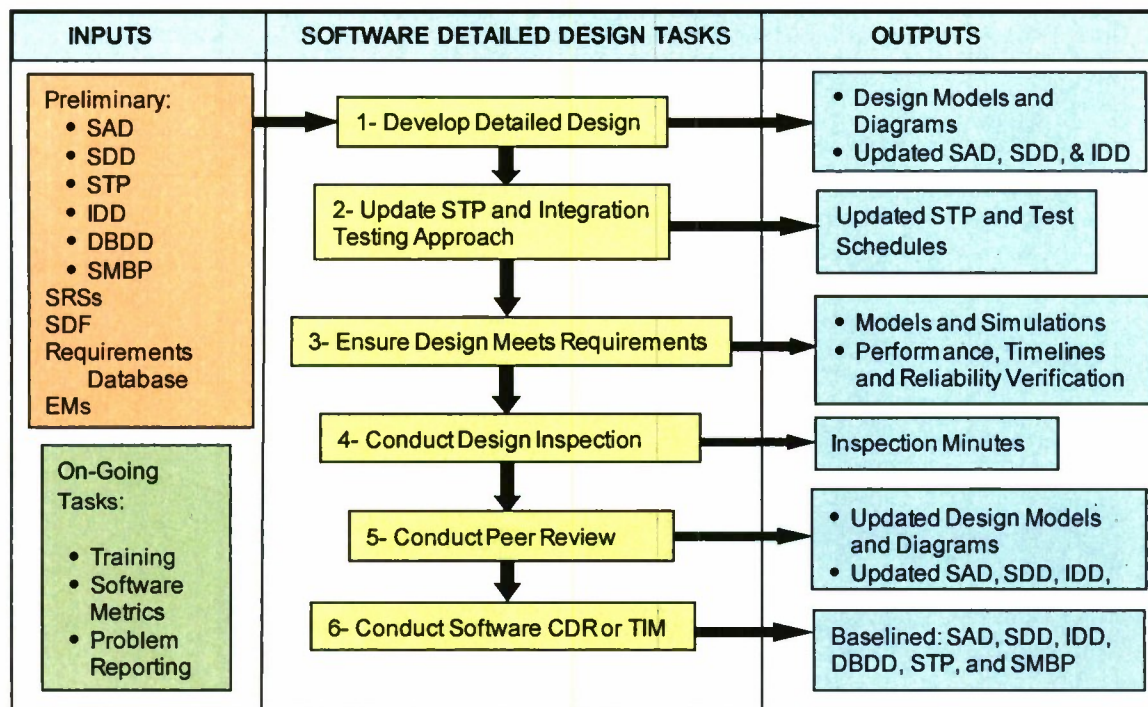
**Objectives.** The objective of SI Detailed Design is to determine the implementation details for each SU. Designers define the specifics of the algorithms or processes an SU is to perform and determine details of the data structures used by the SU internally—and for interactions with other SUs. The resulting SU detail design descriptions are normally sufficient for code developers to implement the design into code. The main objectives are to:

- Complete identification of design components including Software Units
- Complete a description of the design for each SU

- Record all results in the SDF
- Baseline the SAD, SDD, STP, IDD, SMBP, and DBDD (if applicable)

The Detailed Design activity involves decomposing the SUs from the SI Architectural Design (see paragraph 5.6.2) into the lowest level SUs. The design **must** be developed in sufficient detail to map the design to the features of the selected programming language, the target hardware, operating system, and network architecture.

The principal tasks recommended for the Detailed Design Process are described by the example flowchart in Figure 5.6.3 and in the related Task Table 5.6.3 in terms of the inputs and outputs of each task. At the conclusion of this activity, the Detailed Design products must be baselined and placed under software configuration control as described in SDP subsection 5.14.



SDD = Software Design Description  
 STP = Software Test Plan  
 IDD = Interface Design Description  
 STD = Software Test Description  
 SMBP = Software Master Build Plan  
 DBDD = Data Base Design Description

SRS = Software Requirements Specification  
 SDF = Software Development File  
 EM = Engineering Memo  
 TIM = Technical Interface Meeting  
 SAD = Software Architecture Description

Figure 5.6.3. Software Item Detailed Design Process Flow—Example

Major tasks performed during SI Detailed Design should include:

- **Refining the Design Model:** Adding additional details to the design model to accommodate detailed decisions and constructs necessary for implementation
- **Defining Implementation Details:** Refining internal design to add data structures, attribute types, visibility, interfaces and usage mechanisms. Factors to consider include: execution time, memory usage, development time, complexity, maintainability, reusable software and hardware resource utilization. Analysis and modeling may be necessary to determine the best design approach.



- **Generating Class Stubs:** Generate code header files and class stubs based on the object model definitions. Design complex class algorithms or logic
- **Prototyping and Simulations:** Performing prototyping and simulation to validate critical processing areas, mitigate implementation risk, or to identify optimizations
- **Generating and Reviewing Products:** Holding peer reviews on Detailed Design products, and adding the Detailed Design information to the SDD, IDD, and DBDD

Table 5.6.3. Software Item Detailed Design Tasks

Task	Inputs	Subtasks	Outputs
1. Develop Detailed Design	<ul style="list-style-type: none"> <li>• Preliminary SAD, SDD, IDD, STP, SMBP, and DBDD</li> <li>• Baseline SRS</li> <li>• Requirements Database</li> <li>• EMs</li> </ul>	<p><b>Define Detailed Design including:</b></p> <ul style="list-style-type: none"> <li>• Analyze models to identify additional requirements</li> <li>• Define, describe, and decompose SU Detailed Design</li> <li>• Design and develop algorithms, prototypes, control mechanisms, and support services</li> <li>• Determine applicability of COTS/Reuse software</li> <li>• Prepare Design Class Diagrams</li> <li>• Prepare dynamic behavior diagrams showing sequencing of component iterations, states and modes, and transitions</li> <li>• Prepare SDD containing detailed design data</li> <li>• Update IDD with detailed design data</li> <li>• Ensure conformance with architecture</li> <li>• Refine database physical design and the DBDD</li> <li>• Perform resource use analysis of timing and sizing budgets</li> <li>• Review requirements and update the Requirements Database</li> </ul> <p><b>Define Interface Design including:</b></p> <ul style="list-style-type: none"> <li>• Allocate and decompose architecture and user interface requirements to a detailed design level</li> <li>• Define interface design external to the SI and between SUs</li> <li>• Define information flow between SUs</li> <li>• Develop design of user screens</li> <li>• Apply human factor standards to user interface design</li> <li>• Coordinate and review interface design updates</li> </ul>	<ul style="list-style-type: none"> <li>• Design Class Diagrams</li> <li>• Updated SAD and SDD</li> <li>• Updated IDD</li> <li>• SA/SD Work Products such as data flow diagrams and structure charts (if applicable)</li> <li>• Data Dictionary</li> <li>• DBDD</li> </ul>
2. Update STP and Integration Testing Approach	<ul style="list-style-type: none"> <li>• Outputs of Task 1</li> </ul>	<ul style="list-style-type: none"> <li>• Generate test software requirements</li> <li>• Document traceability between software test cases and software test requirements in the STP</li> <li>• Update schedules for conducting each test case</li> <li>• Identify needed integration information (input data, scenarios, data analysis, etc.)</li> </ul>	<ul style="list-style-type: none"> <li>• Updated STP</li> <li>• Test Schedules</li> </ul>
3. Ensure Design Meets Requirements	<ul style="list-style-type: none"> <li>• Outputs of Task 1</li> </ul>	<ul style="list-style-type: none"> <li>• Design software performance and reliability models and develop simulations</li> <li>• Conduct analysis to determine if design meets requirements</li> </ul>	<ul style="list-style-type: none"> <li>• Models and Simulations</li> <li>• Verification of Performance, Timelines and Reliability</li> </ul>
4. Conduct Design inspection	<ul style="list-style-type: none"> <li>• Design Documents</li> <li>• Requirements Database</li> <li>• Design Work Products</li> </ul>	<p>As defined in subsection 5.15 of this SDP Guidebook:</p> <ul style="list-style-type: none"> <li>• Inspect for links of requirements to Detailed Design components</li> <li>• Inspect object-oriented products</li> <li>• Perform document reviews</li> </ul>	<ul style="list-style-type: none"> <li>• Inspection Minutes</li> </ul>
5. Conduct Peer Review	<ul style="list-style-type: none"> <li>• Preliminary SAD, SDD, IDD, STP, DBDD, and SMBP</li> <li>• OO Products</li> <li>• SA/SD Products</li> </ul>	<ul style="list-style-type: none"> <li>• For MC-1 software, segment SI IPT conducts an internal design review for the segment software</li> <li>• For SS-1 and SS-2 software, segment SI IPT conducts an internal segment software TIM</li> <li>• Update design documentation as required</li> </ul>	<ul style="list-style-type: none"> <li>• Updated SAD, SDD, IDD, STP, DBDD &amp; SMBP</li> <li>• Detailed Design Diagrams</li> <li>• Peer Review Minutes</li> </ul>
6. Conduct Software CDR (MC-1) or TIM (SS-1 and SS-2)	<ul style="list-style-type: none"> <li>• Updated Design Documents</li> <li>• Detailed Design Diagrams</li> </ul>	<ul style="list-style-type: none"> <li>• Schedule the CDR/TIM, determine attendees, and update the evaluation criteria</li> <li>• Conduct the CDR/TIM and generate minutes and action items</li> <li>• Ensure closure of action items and generate final outputs</li> </ul>	<ul style="list-style-type: none"> <li>• Baseline SAD, SDD, IDD, STP, SMBP, and DBDD</li> </ul>



Other tasks that may be performed (if applicable) in this activity include:

- Define detailed software user interfaces to the architectural design level and validate it with software prototypes, working models, simulations, and/or display layouts
- Identify concurrency in threads or capabilities
- Identify global resources and determine mechanisms for access control
- Choose the implementation method of control in software (e.g., procedure driven, event driven, or independent tasks)
- Determine methods for handling boundary conditions (i.e., initialization, termination, and failure) and establishing trade-off priorities
- Prepare computer system hardware diagrams including purpose of each component, its interfaces and physical processing characteristics
- Describe how and where the architecture supports Modular Open Software Architecture (MOSA) principles
- Analyze and document the availability of Non-Developmental Items (NDI), incorporate NDI into the design, and allocate requirements to it
- Consider reusable architecture designs for all or portions of a SI; trade-off studies and analyses may be necessary to determine the best design approach
- Draft versions of the Software Users Manual (SUM) and the Software Transition Plan (STrP)

**Approach.** During SI Detailed Design, the designers complete the refinement of the work products (e.g., models and diagrams of the SI). General operations identified in earlier versions of the products **must** be defined to the SU level of functions and procedures, and then defined as to how specific algorithms and support services are implemented in software. This process should occur repeatedly with each build.

Details of the data structures **must** be defined, including temporary data items. The physical database design, if any, **must** also be defined, including data entities, attributes, relationships, and constraints. Interfaces determined in architectural design, including user interfaces, are refined and elaborated. The software Detailed Design tasks **must** refine the software system architecture until the lowest level classes and interfaces have been identified and described.

Detailed Design **must** be performed for each software increment in the current build. There may be multiple builds and design components concurrently in various overlapping stages of completion. For an iterative software lifecycle process, components may have been partially designed during prior software development builds, and only the additional design details for the current build **must** be added.

For MC-1 software only, the Detailed Design activity ends with a formal Critical Design Review (CDR) in which the baselined design documents are evaluated. For software that is developed in multiple builds, only a subset of the SUs may undergo Detailed Design. The SUs that undergo Detailed Design should be only those units necessary to meet the SI requirements for that build, as specified in the Requirements Database.

For SS-1 software, the CDRs are normally replaced with Technical Interface Meetings (TIMs). For SS-2 software, the CDRs are typically peer reviews held for each build. At the conclusion of the Detailed Design activity, all work products **must** be placed under configuration control as described in SDP subsection 5.14.

## 5.7 Software Implementation and Unit Testing

**Introduction.** The Software Implementation and Unit Testing activity of the development lifecycle is often referred to as software Coding and Unit Testing (CUT). The latter acronym will be used in subsection 5.7 to avoid confusing the acronym used for this activity (I&UT) with the Unit Integration and Testing (UI&T) activity described in SDP subsection 5.8. In accordance with TOR-3537B, the Software Implementation and Unit Testing activity **must** be described in five paragraphs in the SDP:

- Software Implementation (paragraph 5.7.1)
- Preparing for Unit Testing (paragraph 5.7.2)
- Performing Unit Testing (paragraph 5.7.3)
- Revision and Retesting (paragraph 5.7.4)
- Analyzing and Recording Unit Test Results (paragraph 5.7.5)

The requirements specified in these sections are for MC-1, SS-1, and SS-2 software only (SS-3 compliance should be optional). There are no C/R requirements in this development activity. Additional products and reviews may be specified in a development's site specific SDP.

**Objectives:** The objective of the Software CUT activity is to convert the SU detailed design into computer code and databases that have been inspected, unit tested, and confirmed. The term *Coding* is used throughout this process to mean the generation of computer-readable instructions and data definitions in a form that can be acted upon by a computer.

**Process Approach.** Major tasks of the Software CUT process include the following:

- The detailed SU design **must** be converted into computer code in accordance with the coding standards for the selected programming language. This may include partial units, or modifications to those created in prior builds
- Specific test descriptions **must** be generated to unit test the SU that define the test cases, test procedures, test input, support data, and expected test results
- The completed source code, test description data on all developed units, and documentation, should be reviewed through a Peer Review inspection, which may include the participation of SQA, prior to execution of the test
- The test cases should be executed against the executable code to determine the success of the coding effort. White Box (structural) and Black Box (functional) tests should be performed on the individual units. Successful completion of unit level testing is a prerequisite for promotion of units to software integration
- The results of the test cases **must** be reviewed and the code reworked and retested until all unit tests have been successfully completed
- The test results **must** be independently reviewed by someone other than the developer to confirm successful completion of the test and that the test data and results have been recorded in the SDF

These steps are highly iterative, in that the code and test tasks are performed for each SU (class) designated to participate in the threads allocated to the current software build. Groups of SUs may be coded, reviewed, and tested as a set, according to the development plan and schedule for the increment. SUs may also be incomplete, in that only the functionality required to support the current increment is implemented.



The Software CUT activity for a single SU formally ends upon completing confirmation of the test results and recording of the test data and results in the SDF. After these actions have been completed, the SU **must** be brought under configuration control. All changes to the SU thereafter **must** be handled using SCRs or SDRs.

Table 5.7-1 summarizes the readiness criteria in terms of entry and exit criteria, confirmation criteria to ensure completion of the required tasks, and the measurements usually collected during the software CUT activity.

Table 5.7-1. Readiness Criteria: Software Coding and Unit Testing—Example

Entry Criteria	Exit Criteria
<ul style="list-style-type: none"> <li>• SU detailed design has been completed.</li> <li>• Software coding standards have been established.</li> <li>• The Software Engineering Environment (SEE) has been established.</li> <li>• The Requirements Test and Verification Matrix (RTVM) and Software Requirements Traceability Matrix (SRTM) is available.</li> <li>• The SDL has been established.</li> </ul>	<ul style="list-style-type: none"> <li>• Software Unit Test Cases have been completed and accepted by the Software Lead.</li> <li>• SU test procedure data has been recorded in the SDF.</li> <li>• SU Source Code has been developed, compiled, debugged, and accepted by the Software Lead.</li> <li>• SU test results have been recorded in the SDF.</li> <li>• Software is put under software integration CM control.</li> </ul>
Verification Criteria	
<ul style="list-style-type: none"> <li>• Peer Reviews of SU Test Cases, SU test data, Source Code and SU test results are completed and recorded.</li> <li>• Software Development Librarian accepts the source code.</li> <li>• SQA performs process/product audits for ongoing product engineering tasks per SDP subsection 5.16.</li> </ul>	
Measurements	
<ul style="list-style-type: none"> <li>• Actual KSLOC coded versus KSLOC planned</li> <li>• Unit testing planned versus actual test progress</li> <li>• Number of defects found in Peer Reviews</li> <li>• SCRs and SDRs opened versus closed</li> </ul>	
(see subsection 5.20)	

**Software Work Products.** The principal product produced during this activity is SU source code as shown in example Table 5.7-2. The format for the source code is established by the coding standards for the particular language used. During the software CUT activity, SU Test Cases, test procedure data, sizing and timing are prepared and updated. The SRS, STP, IDD, and DBDD may also be updated as required. In addition, during the CUT activity, draft versions (if applicable) of the Computer Programming Manual (CPM as described in subparagraph 5.13.8.1) and the Firmware Support Manual (FSM as described in subparagraph 5.13.8.2) may be prepared.

Table 5.7-2. Required Software Coding and Unit Testing Work Products—Example

Software CUT Products	MC-1	SS1	SS2	SS3
Source code or reference to source code in SDFs	Required	Required	Required	Optional
SU Test Cases, procedures, data and test results	Required	Required	Required	Optional

**Roles and Responsibilities.** Table 5.7-3 is an example of the roles and responsibilities of the Software IPT personnel, software developers, and other groups during the software CUT activity.



Table 5.7-3. Roles and Responsibilities During Software Coding and Unit Testing—Example

Roles	Responsibilities
<b>Software IPT and Developers</b>	Codes SUs to the appropriate coding standards
	Develops the unit test description and executes the unit test
	Conducts the required inspection of the source code and test documentation
	Reworks and retests the SU when problems are identified
	Arranges for confirmation of the test results per the Software Reviews Standards (The SU author participates in the inspection, but someone other than the author performs the inspection and confirmation of the test results)
	Updates and maintains the SDF with source code, unit test descriptions, test code, and unit test results
	Ensures critical software requirements are traced to SUs
	Collects and reports software CUT task metrics
<b>Software Test Personnel</b>	Continues development of STP
<b>CSWE</b>	Reviews the SDFs, reviews task products, and attends activity reviews
<b>SQA</b>	Evaluates the Segment IPT software products for adherence to documented policies and procedures, evaluates Segment IPT products for product quality, and reports findings to upper-level management
<b>CCB</b>	Addresses segment SCR/SDRs as they are generated
<b>SWCCB</b>	Addresses SU SCR/SDRs involving external interface changes or SU changes from prior build releases as they are generated by the Segment IPT software personnel
<b>SCM</b>	Processes all SCR/SDRs for SU changes to the source code and test documentation

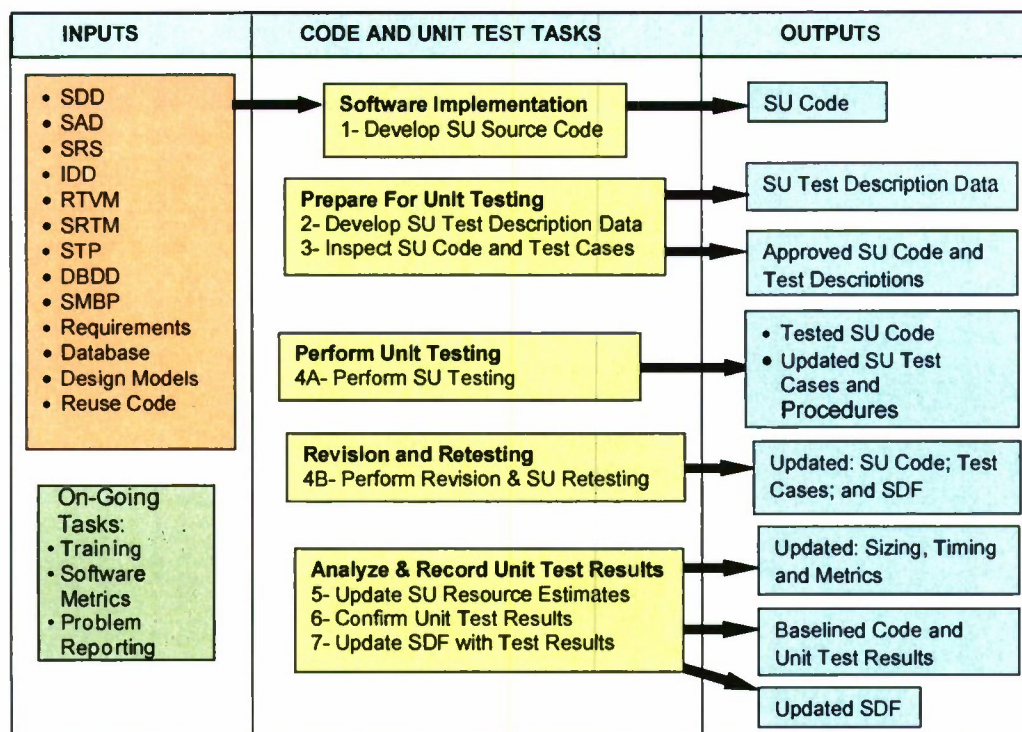
**Process Tasks.** An example of the Software CUT process is shown in Figure 5.7. Since this is an iterative process, there are no vertical arrows in the process flow chart. The seven process tasks are expanded with more detail in its related Task Table 5.7-4 containing the inputs and outputs to each task.

### 5.7.1 Software Implementation

**Objectives.** The objective of software implementation is to implement requirements by converting the software unit detailed design into source code and from the source code generate executable computer code. The major tasks that **must** be accomplished are:

- Develop the code for each SU based upon the design requirements and detailed design
- Code the software using the required coding standards
- Create executable code and debug using applicable tools
- Update source code estimates with actual measurements of the SU
- Document decision to reuse code and identify reuse code modules

**Approach.** The software developers **must** generate the source code using the appropriate programming language for each SU, based on the detailed design, interface requirements, and supporting design information. Changes made to the executable code **must** be accomplished through modification of the source code and subsequent recompilation or reassembly. Once successfully compiled and executed, a Peer Review inspection of the source code should be performed.



**NOTE:** There are no vertical flow arrows since this is an iterative process

SDD = Software Design Description

RTVM = Requirements Test Verification Matrix

SU = Software Unit

IDD = Interface Design Description

STP = Software Test Plan

SDF = Software Development Files/Folders

SRTM = Software Requirements Traceability Matrix

SRS = Software Requirements Specification

SAD = Software Architecture Description

SMBP = Software Master Build Plan

DBDD = Data Base Design Description

Figure 5.7. Software Coding and Unit Testing Process Flow—Example

Table 5.7-4. Software Coding and Unit Testing Tasks—Example

Task	Inputs	Subtasks	Outputs
1. Develop SU Source Code	<ul style="list-style-type: none"> <li>See Figure 5.7</li> </ul>	<ul style="list-style-type: none"> <li>Check SUs against input documents, SCRs, and reused code to confirm definitions and requirements of SUs</li> <li>Code SUs</li> </ul>	<ul style="list-style-type: none"> <li>SU Code</li> </ul>
2. Develop SU Test Description Data	<ul style="list-style-type: none"> <li>SU Code</li> </ul>	<ul style="list-style-type: none"> <li>Address SU requirements in Test Cases</li> <li>Develop test inputs and outputs</li> <li>Develop SU Test Cases and test exceptions</li> <li>Ensure adequate Test coverage and number of iterations</li> </ul>	<ul style="list-style-type: none"> <li>SU Test Cases</li> </ul>
3. Inspect SU Code and Test Cases	<ul style="list-style-type: none"> <li>SU Source Code</li> <li>SU Test Cases</li> </ul>	<ul style="list-style-type: none"> <li>Schedule SU for Peer Review after successful compilation</li> <li>Hold Peer Review inspection of SU source code and Test Cases</li> <li>Close Peer Review findings</li> </ul>	<ul style="list-style-type: none"> <li>Approved SU Code and Test Description Data</li> </ul>
4. Perform SU Testing, Revision and Retesting	<ul style="list-style-type: none"> <li>SU Code</li> <li>SU Test Cases</li> </ul>	<ul style="list-style-type: none"> <li>Follow procedures in SU Test Cases</li> <li>Record test results in the SUs SDF</li> <li>Fix source code problems</li> <li>Modify and approve unit test procedures and results</li> <li>Retest; repeat until unit testing is successful</li> </ul>	Updates to: <ul style="list-style-type: none"> <li>Tested Code</li> <li>SU Test Cases and Procedures</li> <li>Segment SDF</li> </ul>
5. Update SU Resource Estimates	<ul style="list-style-type: none"> <li>Tested Source Code</li> </ul>	<ul style="list-style-type: none"> <li>Measure sizing, timing and complexity of SU as required</li> <li>Record SU SLOC count and productivity metrics in the SDF</li> <li>Update metrics with measurements from SU testing</li> </ul>	<ul style="list-style-type: none"> <li>Sizing and Timing in SDF</li> <li>Updated Metrics</li> </ul>



Task	Inputs	Subtasks	Outputs
6. Confirm Unit Test Results	<ul style="list-style-type: none"> <li>• SU Test Results</li> </ul>	<ul style="list-style-type: none"> <li>• Verify correctness of test results</li> <li>• Capture test procedures, inspection results, and unit test results</li> <li>• Place tested code under configuration control</li> <li>• Update SRS as required</li> <li>• Release SU for Unit Integration</li> </ul>	<ul style="list-style-type: none"> <li>• Baseline Source Code</li> <li>• Updated SRS as required</li> </ul>
7. Update SDF with Test Results	<ul style="list-style-type: none"> <li>• Update information from 1–6 above</li> </ul>	<ul style="list-style-type: none"> <li>• Update SDF and document Lessons Learned</li> <li>• Record test results in the SDF</li> <li>• Prepare Release Notice to inform availability of SUs</li> </ul>	<ul style="list-style-type: none"> <li>• Updated SDF</li> <li>• Release Notice</li> </ul>

### 5.7.2 Preparing for Unit Testing

**Objectives.** Objectives of the Preparing For Unit Testing is to:

- Develop overall test objectives and assumptions including constraints
- Define, develop, and document the Unit Test Cases and Unit Test Procedures
- Develop input test data including data files, databases, algorithm, and simulation data
- Identify support resources, including required drivers and stubs
- Test preparation (including hardware and software)
- Describe the inputs, expected results, success criteria, and evaluation criteria for each test case
- Allocate software requirements to each test case and ensure that all SU requirements are tested
- Define data files, databases, simulation programs, and additional resources required
- Layout a preliminary schedule of when the unit test cases are to be performed
- Execute all statements and branches of the software unit at least once
- Identify and define interfaces and dependencies between the test cases
- Identify start-up, termination, restart, error and exception handling procedures
- Verify that the software unit performs its intended operations using nominal and boundary upper and lower limit input values
- Record the above information in the SDF

**Approach.** The software developers **must** identify test cases and procedures to be performed on a software unit. For cases where tests cannot be developed to adequately verify that functionality has been demonstrated, verification by analysis may be permitted. This situation can arise where: (a) an event to be tested is difficult to cause or, (b) involves prohibitively extensive testing or cost. Modified reused SUs require complete re-testing of the software unit. If the reused SU is deemed critical it **must** be unit tested even if it has not been modified. The completed test description, including both test case definitions and test procedures, **must** be retained in the SDF. The inspection package for modified reused code should also include a code difference listing.

### 5.7.3 Performing Unit Testing

**Objectives.** The objectives of performing unit testing are to:

- Perform unit testing of the developed source code in accordance with the unit test cases and test procedures
- Verify the unit level functional, interface, and SU performance requirements
- Verify the SUs exception handling capability



- Maintain unit test logs to verify and track SU test execution and completion
- Update the unit source code to correct errors detected during the unit testing
- Record the unit test results and performance measures in the SDF

**Approach.** Software testers normally begin by ensuring that all necessary data, tools, test environment, and unit test configuration are available. When all required pieces for the test are assembled, the test can proceed per the test procedures. The testers **must** verify the unit level functional, interface, and performance requirements. The testers **must** collect and record the test outputs, logs, notes, results, and discrepancies found. Although software developers **must** ensure each SU satisfies its requirements, unit testing may be considered principally “white box” testing, i.e., testing against the design.

#### 5.7.4 Unit Testing Revision and Retesting

**Objectives.** The objectives of revision and retesting are to: (a) modify/rework the source code and test description to eliminate any problems identified during unit testing; and (b) retest the unit to verify that the changes have been successful and have not produced side effects. If a unit test fails, the problem **must** be fixed and the test(s) repeated. The standard design inspection process **must** be invoked again and the SDFs updated. Regression testing of affected SU test cases **must** be performed after any modification to previously tested software. Changes **must** be made in accordance with the Corrective Action Process (see SDP subsection 5.17).

**Approach.** Test results, and the documented problems, **must** be evaluated by software developers to identify needed changes to the SU and test description. This task should be repeated until all the SU test cases have been successfully completed.

#### 5.7.5 Analyzing and Recording Unit Test Results

**Objectives.** The objective of analyzing and recording unit test results is to finalize the unit testing for a SU by ensuring that:

- The unit satisfies the expected results of the test cases
- The test data, test results, unit test dependencies, and supporting analysis material have been recorded in the SDF
- Root cause analysis of problems has been performed
- The SU is ready to be released for Unit Integration and Testing (see SDP subsection 5.8)

**Approach.** After completing unit testing, the Software Lead **must** perform an independent confirmation of the test results and ensure the results have been recorded in the SDF. If discrepancies or problems are found, then appropriate corrective actions **must** be performed.

Once the independent review signifies that the SU has successfully passed the verification process, the SU can be baselined and brought under configuration control. The SU source code is then submitted for incorporation into software integration builds. In addition, developers should incorporate supporting analysis material and unit test dependencies information in the appropriate SDF. The related metrics measurements obtained during SU testing should also be updated.

### 5.8 Unit Integration and Testing

**Introduction.** Subsection 5.8 of the SDP addresses the objectives, approach, readiness criteria, software work products, roles and responsibilities, and tasks specific to the software Unit Integration

and Testing (UI&T) activity of the development process. In accordance with TOR-3537B, the software Unit Integration and Testing activity **must** be described in four paragraphs in the SDP:

- Prepare for UI&T—including updating the STP and test procedure data (paragraph 5.8.1)
- Perform UI&T—including performing the integration and test of a build in accordance with integration test procedures (paragraph 5.8.2)
- Revision and Retesting—including reworking the source code; perform regression testing for changes occurring during UI&T and documenting the discrepancies (paragraph 5.8.3)
- Analyze and Record UI&T Results—including analyzing test results, documenting the software UI&T results in the SDF, and identifying who decides an Integration Build is ready for release to SI Qualification Testing (paragraph 5.8.4)

**UI&T Objectives.** The objective of the UI&T activity is to perform a systematic and iterative series of integration builds on Software Units (SU) that have successfully completed Code and Unit Test, and build them up to a higher level SU (formerly called a Software Component), or Software Item (SI), for the current build. The Software Test Plan (STP) and software test procedure data should be reviewed for consistency with the Software Master Build Plan (SMBP), and revised if necessary. In addition, preparation of a draft Software Version Description (SVD) and a draft of the Software Test Description (STD) for qualification testing (see SDP subsection 5.9) should begin during this activity.

Segment software integration teams **must** develop the integration plans, integration test cases, and integration test procedures and test data in preparation for the actual integration and test. The SUs should be checked out of the controlled area of the Software Development Library (SDL) by the integrators. As the builds are successfully integrated, the SUs are typically returned to the SDL to be elevated to a higher level of control. Discrepancies **must** be recorded on SDRs.

**UI&T Approach.** The UI&T activity consists of the following major activities:

- The software integration plans, test cases, and test procedures **must** be developed and peer reviewed
- Test data, tools, drivers, simulators, etc. **must** be in place before start of testing
- The integration test procedures **must** be executed against the executable code
- Needed corrections to the software, and the integration test procedures, **must** be made and the affected integration iteration retested; this activity should be repeated until all SUs have been successfully integrated and have met the test acceptance criteria
- Test results for integrated SIs **must** be independently analyzed, or with a Peer Review, to verify successful integration and recording of results in the Software Development File (SDF)
- The SI STP should have been baselined prior to start of actual testing
- Regression testing **must** be performed as needed to incorporate SUs from prior builds

The UI&T activity formally ends with the verification of the test results and the recording of the test data and test results in the SDF. The build **must then be baselined and moved to the verification area of the SDL. All changes to a SI thereafter must be handled through the process described in the SCM Plan.**

**UI&T Readiness Criteria.** Table 5.8-1 summarizes the readiness criteria in terms of entry and exit criteria, verification criteria to ensure completion of the required activities, and the required measurements normally collected during the Software UI&T activity.



Table 5.8-1. Readiness Criteria: Software Unit Integration and Testing—Example

Entry Criteria	Exit Criteria
<ul style="list-style-type: none"> <li>• Software Test Plan (STP) is available.</li> <li>• Coding and testing of the SUs have been completed.</li> <li>• Software test procedures data is available.</li> <li>• Integration builds are available from the SDL.</li> <li>• The RTVM, SRTM, and SMBP are available.</li> </ul>	<ul style="list-style-type: none"> <li>• SI build is successfully integrated, accepted by the Software team lead, and turned over to the SDL.</li> <li>• The draft Software Version Description (SVD) is approved by the Software Team Lead.</li> <li>• The STP is updated and ready to support SIQT.</li> </ul>
Verification Criteria	
<ul style="list-style-type: none"> <li>• Software Peer Reviews have been successfully completed.</li> <li>• Unit integration plans, test cases, and UI&amp;T procedures developed and successfully peer reviewed.</li> <li>• Software Units successfully integrated in accordance with the integration plans.</li> <li>• The Software Team Lead reviews and approves the integration test reports and integration release notice.</li> <li>• All SUs in the build per the SMBP are successfully integrated and tested and the results stored in SDFs.</li> <li>• SQA performs process/product audits for ongoing product engineering activities per SDP subsection 5.16.</li> </ul>	
Measurements	
<ul style="list-style-type: none"> <li>• Defects found from Peer Reviews</li> <li>• SDRs opened versus closed</li> <li>• Units integrated—planned versus actual</li> <li>• SLOC count—planned versus actual</li> </ul>	
(see subsection 5.20)	

**UI&T Software Work Products.** Examples of software work products for the UI&T activity are summarized in Table 5.8-2.

Table 5.8-2. Software UI&amp;T Work Products—Example

Software UI&T Work Products per Build	MC-1	SS-1	SS-2	C/R
Updated STP	Required	Required	Required*	Required*
Draft Software Version Description (SVD)	Required	Required	Required*	Required*
Draft Software Test Description (STD)	Required	Required	Required*	Required*
SI test cases traced to SRS requirements in the requirements database	Required	Required	Required*	Optional
UI&T Products: <ul style="list-style-type: none"> <li>• Unit integration plans</li> <li>• UI&amp;T test cases and procedures</li> <li>• UI&amp;T scripts, drivers and test data</li> <li>• UI&amp;T test results</li> </ul>	Required*	Required*	Required*	Required*
*Document not required, but applicable information is developed and retained in the SDF.				

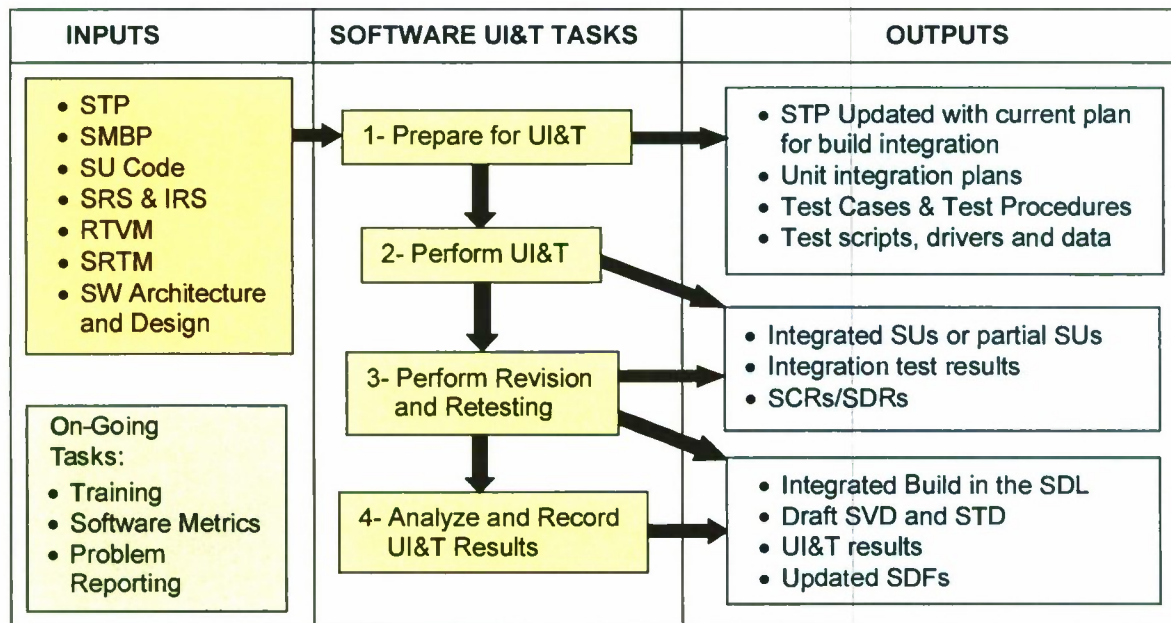
**Roles and Responsibilities.** Software developers **must** be responsible for development of the integration test plans, procedures, data, actual integration of the SUs, and the execution of the tests. When problems are identified, the software developers **must** be responsible for reworking SUs and retesting the integration of those units. Table 5.8-3 summarizes typical responsibilities and roles for the UI&T activity.



Table 5.8-3. Software UI&amp;T Responsibilities—Example

Group Roles	Responsibilities	
Software Development Personnel	Conducts the required peer reviews of the UI&T documentation	
	Updates and maintaining the SDF with test procedures and test results	
	Address safety, security, privacy and other critical software requirements in the integration test cases	
	Collects and reports SU integration and testing activity metrics	
	Submits SDRs as necessary	
Other Groups	Software Test	Update the STP (as needed) and prepare drafts of the STD and SVD
	Chief Software Engineer (CSWE)	Reviews the SDFs, reviews the activity products, attends activity reviews, and monitors and analyzes software metrics
	SQA	Evaluates the software IPT for adherence to the documented policies and procedures, evaluates software IPT products for product quality, witnesses testing and documents, and reports findings to upper-level management
	Segment CCB	Addresses internal SCR/SDRs as they are generated
	Software/CCB	Addresses SU SDRs involving external interface changes and prior build release SU changes as they are generated by the software IPTs
	SCM	Processes SDRs for SU changes to the source code and test documentation

**UI&T Process Activities.** The Software UI&T process is shown in Figure 5.8. The four process tasks are also shown expanded with more details in its related Task Table 5.8-4 containing the inputs and outputs to each UI&T task.



STP = Software Test Plan

RTVM = Requirements Test Verification Matrix

SMBP = Software Master Build Plan

SRS = Software Requirements Specification

SCR/SDR = Software Change Request/Software Discrepancy Report

SVD = Software Version Description

SDF = Software Development Files

SDL = Software Development Laboratory

IRS = Interface Requirements Specification

Figure 5.8. Software UI&amp;T Process Flow—Example

Table 5.8-4. Software UI&amp;T Tasks—Example

Tasks	Inputs	Subtasks	Outputs
1. Prepare for UI&T	<ul style="list-style-type: none"> <li>See input documents in Figure 5.8</li> </ul>	<ul style="list-style-type: none"> <li>Update STP with current plan for build integration</li> <li>Link requirements to integration and test cases</li> <li>Create and populate the traceability database</li> <li>Prepare Unit integration plans, UI&amp;T test cases, procedures, scripts, drivers, and test data</li> </ul>	<ul style="list-style-type: none"> <li>Updated STP</li> <li>UI&amp;T Products</li> </ul>
2. Perform SU Integration and Testing	<ul style="list-style-type: none"> <li>SU code</li> <li>Test Cases</li> <li>Test Data</li> <li>Test Scripts and Drivers</li> <li>Test Procedures</li> </ul>	<ul style="list-style-type: none"> <li>Integrate the SUs per the integration plans</li> <li>Conduct integration testing of the current build based on the test cases</li> <li>Record and document anomalies and errors detected during testing in the Software Discrepancy Report (SDR)</li> </ul>	<ul style="list-style-type: none"> <li>Integrated SUs or Partial SU Build</li> <li>SDRs</li> <li>Test Results</li> </ul>
3. Perform Revision and Retesting	<ul style="list-style-type: none"> <li>Integrated SUs</li> <li>Updated SUs to fix SDRs</li> <li>Test Results</li> </ul>	<ul style="list-style-type: none"> <li>Perform regression testing to accommodate new functions or changes to the previously integrated code</li> <li>Perform retesting after fixes to test procedures and/or code</li> <li>Record and document anomalies and errors detected during testing in the SDR</li> </ul>	<ul style="list-style-type: none"> <li>SDRs</li> <li>Test Results</li> </ul>
4. Analyze and Record UI&T Results	<ul style="list-style-type: none"> <li>SDRs</li> <li>Test Results</li> </ul>	<ul style="list-style-type: none"> <li>Analyze results and document findings of the integration tests in the SDF</li> <li>Inform software development and the SDL that the current build has successfully completed integration testing</li> <li>Prepare Draft SVD and STD</li> <li>Conduct Independent review of test results</li> </ul>	<ul style="list-style-type: none"> <li>Integrated Build in SDL</li> <li>SVD and STD Draft</li> <li>Updated SDF</li> <li>Review Completed</li> </ul>

### 5.8.1 Preparing for UI&T

**Objectives.** The principal objective of preparing for the UI&T task is to establish test cases, test procedures, and test data for conducting unit integration and testing to define a systematic and iterative approach for integrating a subset of SUs until the entire set of SUs are integrated into the complete SI (for that build).

As a minimum, the test cases **must** cover a description of:

- Execution of all interfaces between software units—including limit and boundary conditions
- Integrated error and exception handling across the SUs under test
- End-to-end functional capabilities through the SUs under test
- All software requirements allocated to the SUs under test
- Performance testing—including operational input and output data rates and timing and accuracy requirements
- Stress testing—including worst-case scenarios
- Start-up, termination, and restart
- Fault detection, isolation, and recovery handling
- Resource utilization measurement

Whenever possible, SU integration should be performed on the target hardware in a configuration as close as possible to the operational configuration. All COTS/Reuse software, whether modified or unmodified, **must undergo software UI&T**. Software developers **must** define integration test cases that are mapped to use cases, and then mapped to requirements and corresponding test procedures, in an integration test description to verify success of each partial integration before proceeding to the next iteration. The specified integration sequences are to: (a) Verify that the SUs operate together

using nominal and exception conditions; and (b) Exercise all interfaces for the SUs that have been integrated.

### 5.8.2 Performing UI&T

**Objectives.** The principal objectives of performing UI&T are to:

- Integrate and combine SUs
- Execute the integration plan and corresponding test procedures as documented in the integration test description to produce the integrated SI
- Execute the integration runs and verify the complete integration
- Verify that SUs within the SI provide the functionality required for that build
- Record test results for this level of testing

During this task, coded and tested SUs should be integrated into SIs by a software integration team in a series of integration builds. The SUs should be obtained from the development controlled area of the SDL. The software personnel performing the integration usually begin by ensuring that all the SUs to be integrated and all necessary data and tools are available. The integration build to be tested should be generated by the developers using baselined SUs obtained from the SDL.

When all required pieces for the integration are assembled, the integration should proceed per the procedures specified in the test description. During integration and testing, the software developers collect or record the outputs, logs, test notes, and test results. All problems, errors, and discrepancies **must** be recorded as SDRs.

### 5.8.3 UI&T Revision and Retesting

**Objectives.** The primary objectives of UI&T revision and retesting are to:

- Revise the source code and regression test in response to problems identified in SDRs, first at the SU level and then at a combined SU level to ensure that existing functionality has not been impaired
- Perform regression tests, as required, to accommodate new functions or changes in the current build
- Document and track integration problems and test errors

The integration team **must** perform the integration tests, and any necessary regression testing and record discrepancies on SDRs which are placed into the Corrective Action process for disposition and rework. The documented problems **must** be evaluated by the software developers to determine the necessary changes to SUs or test descriptions. In cases where SUs require changes, SDRs (and Software Change Requests if used) **must** be generated and the changes handled by the corrective action process.

In cases where test descriptions require modification, the appropriate changes **must** be made and a version history included in the test description. Retesting **must** also be performed when test procedures are changed. **Retesting must be repeated as needed until all SUs have been successfully integrated and tested.**

Software personnel determine the necessary modifications to source code, SDFs, and/or documentation. Source code and documentation are modified based on approved changes by the



SWCCB (typically at the SU Level) and at the segment's CCB or equivalent (typically at the SI level). Changes **must** be handled with SDRs in accordance with subsection 5.17.

#### 5.8.4 Analyzing and Recording UI&T Results

**Objectives.** The primary objectives of analyzing and recording UI&T results are to:

- Verify that the tests have been successfully completed and that the test data and results have been recorded in the SDF
- Handle changes to SIs after being brought under SCM control using SDRs
- Complete SI level integration by the successful execution of all of the defined integration test procedure runs
- Meet integration completion criteria and perform root cause analysis for deficiencies
- Document SI level integration, and modify SDFs, source code, and documentation
- Prepare, analyze, and document results of the integration tests, and place in the appropriate SDF location

Upon successful completion of a round of integration, the Software Lead should authorize the release of the build by providing the Software Development Librarian with a release notice. The integration team **must** document the results of each round of integration and place it in the appropriate location in the SDF.

After all SUs have been successfully integrated and tested, the software IPT Lead should perform the independent review of the test results and ascertain that the integration test data and results have been recorded in the SDF. Peer Reviews may also be used.

The SI can then be submitted to the SCM Librarian and baselined in the SDL Verification Area in preparation SI Qualification Testing. Specific procedures for recording, analyzing, verifying, and storing UI&T results should be included in the SDP.

### 5.9 Software Item Qualification Testing

**Introduction.** This subsection of the SDP addresses the objectives, approach, documentation, staff responsibilities, and tasks for the Software Item Qualification Testing (SIQT) activity. In accordance with TOR-3537B, the SIQT activity **must** be described in seven paragraphs in the SDP; the last five of these tasks are the sequential processing steps of the SIQT activity:

- Independence in Software Item Qualification Testing (paragraph 5.9.1)
- Testing on the Target Computer System (paragraph 5.9.2)
- Preparing for SIQT by preparing the SIQT Software Test Description(s) (STD) and updating the Software Test Plan (STP) (paragraph 5.9.3)
- Performing a dry run of the SI qualification test procedures on the target computer hardware to ensure that the tests and test descriptions are complete and accurate and ready for the formal SIQT witnessed testing (paragraph 5.9.4)
- Performing Software Item Qualification Testing by executing the formal SIQT using the STD and recording the test results, (paragraph 5.9.5)

- Revision and Retesting including implementing all necessary corrections to the software products and the software test procedures in the STD, then retesting the tests that failed and any others involving modified SUs and their interfaces. This activity is repeated until all SI tests have been completed (paragraph 5.9.6)
- Analyzing and Recording SIQT Results and anomalies in a Software Test Report (STR) (paragraph 5.9.7)

**Objectives.** The objective of SIQT is to demonstrate that the SI meets the software and interface requirements allocated to the SI. **SIQT must be a controlled and documented activity assigned to software test engineers who are independent of the software development team.** SIQT must demonstrate that: the software performs correctly; contains the features prescribed by its requirements at the SI level; and properly interacts and performs its specified functions within the total system as documented in the SRS for each build.

**Approach.** For software developed in multiple builds, the SIQT for each build **must** address the software and interface requirements allocated to the current build being tested and the SIQT for the SI being tested will not be completed until the final build for that SI. Regression tests **must** be performed as needed throughout the iterative process. **The software test results must be documented after each test.**

A Software Test Report (STR) **must** be published to document the final test results. Any discrepancies noted **must** be recorded in Software Discrepancy Reports (SDRs), analyzed, and dispositioned in accordance with the Corrective Action Process. If the corrections are deferred for a future release, then the STR, and all related release documentation (e.g., the Version Description Document), **must** reflect SI constraints or work-arounds needed.

The activity ends when documentation of the software test results is completed and open SCRs/SDRs that can be resolved are resolved for the current release. All test materials and results **must** be “impounded” to establish the “as conducted” archive. A post-test debrief should be conducted to evaluate preliminary results, to analyze anomalies that occurred and to collect lessons learned. Table 5.9-1 summarizes the readiness criteria in terms of entry and exit criteria, verification criteria to ensure completion of the required activities, and the measurements normally collected during the SIQT activity.

Table 5.9-1. Readiness Criteria: Software Item Qualification Testing—Example

Entry Criteria	Exit Criteria
<ul style="list-style-type: none"> <li>• SRS, IFCD, SMBP, SDD, SAD, IDD, DBDD, and STP have been baselined.</li> <li>• The Requirements Traceability Verification Matrix (RTVM) is available.</li> </ul>	<ul style="list-style-type: none"> <li>• Formal Software Item qualification tests are successfully completed including an action plan generated to close remaining SDRs.</li> <li>• STDs and STRs are completed.</li> <li>• STP is updated as required.</li> </ul>
Verification Criteria	
<ul style="list-style-type: none"> <li>• Software Peer Reviews have been successfully completed for all required documentation.</li> <li>• SQA and customer witness test execution</li> <li>• SQA performs process/product audits for ongoing product engineering activities per SDP subsection 5.16</li> </ul>	
Measurements	
<ul style="list-style-type: none"> <li>• Number of Peer Review defects</li> <li>• Number of SCRs and SDRs opened and closed, aging data, origin, and root cause of problems</li> <li>• Number of test cases completed and number of requirements verified</li> </ul>	

(see subsection 5.20)

If SCR or SDR fixes are incorporated into the software, then portions of the SIQT test procedures **must** be re-run to verify that applicable SCR/SDR fixes are implemented and working correctly. In



addition, it **must** be determined that selected pre-existing functionality is still performing per software and interface requirements after the fixes have been implemented. SDP paragraph 5.9.6 covers details on performing revision and re-testing activities.

**Work Products.** Documentation products normally produced during the SIQT activity for each SI include: Software Test Description (STD), Software Test Report(s) (STR), an updated Software Test Plan (STP), and traceability products from the Requirements Database. For MC-1 and SS-1 software these software products **must** be documented in the STP, STD, and STR for each SI and the traceability products contained in the Requirements Database. For SS-2 software, the SI test description and test results may be documented in the SDF. Annex-H of J-16 describes the recommended format and contents of the STD and STR and they can be summarized as follows:

- The STD describes the SI-specific test cases and corresponding software and interface requirements, test environment, test procedures, input data, simulations or emulations, expected results, and success criteria
- The STR specifies or references the test outputs, logs, notes, and test results

If the SI is reused “as is” and the existing documentation meets the minimum requirements, the existing documentation can be used “as is” also. Work products for this activity are summarized in Table 5.9-2. The documentation **must** be made available via an electronic data repository system. Test Logs, describing the results of the tests, are not listed, but are required.

Table 5.9-2. Software Item Qualification Testing Work Products Per Build—Example

SIQT Documentation	MC-1	SS-1	SS-2	S-3	C/R
STP and STD (separate document per SI)	Required	Required	Required*	Optional	Optional
SI test description and test results in SDF	Required	Required	Required	Required	Required
STR	Required	Required	Required*	Optional	Optional
SI test cases traced to SRS requirements in the requirements databases	Required	Required	Required*	Optional	Optional

\*Document not required but applicable information is developed and retained in the SDF.

In addition, preliminary versions of the following software documents (if applicable) may be prepared concurrently with the SIQT activity:

- Software Product Specification (SPS as described in paragraph 5.12.1)
- Software Version Description (SVD as described in paragraph 5.12.2)
- Software Users Manual (SUM as described in subparagraph 5.12.3.1)
- Computer Programming Manual (CPM as described in subparagraph 5.13.8.1)
- Firmware Support Manual (FSM as described in subparagraph 5.13.8.2)
- Software Transition Plan (STrP as described in paragraph 5.13.9)

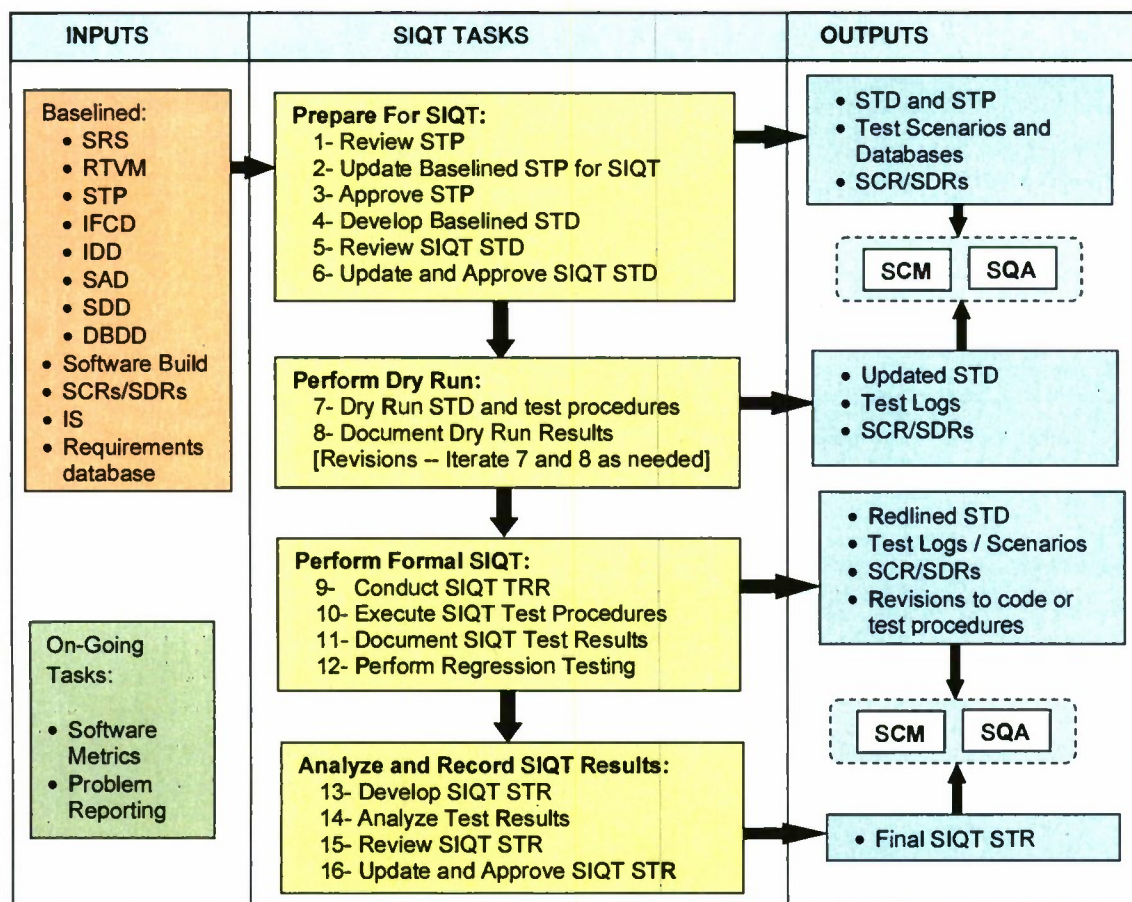
**Roles and Responsibilities.** The software test lead, supported by software test personnel, should be responsible for the development of the SI test plan and test description. Execution of the SI test should be performed by the software test engineers. For support software, the software test lead may have software developers develop and run the tests, provided they are not the same individuals who performed SI integration. Where problems are identified during SI testing, software developers should work with test engineers to analyze problems to determine if it is a software issue or a test procedure issue. Table 5.9-3 is a summary of responsibilities for software developers and roles of other groups in the SIQT activity.



Table 5.9-3. SIQT Roles and Responsibilities—Example

Roles	Responsibilities
Software Development	Implements software changes as a result of SCR/SDRs
	Addresses critical software requirements in the software test cases
	Generates traceability products for SI test cases to SRS requirements Program Unique Identifier (PUI)
	Collects and reports software metrics; Updates and maintains the SDF
CSWE	Monitors SI tests, reviews SDFs, reviews software products, attends reviews (or designee), and monitors and analyzes software metrics. The CSWE must concur that the SI is ready for qualification testing
Software Test	Executes the SI test and submits SCR/SDRs as necessary for detected problems
System Safety	Supports verification, testing, and tracking of safety-critical requirements
SQA	Witnesses program and/or Element SQA and audits the tests, attends reviews and inspections, evaluates adherence to documented policies and procedures, monitors the quality of output products, and documents and reports findings to management
Element CCB	Addresses all current build factory/segment internal SCR/SDRs, as they are generated by the IPT software personnel or software test engineers
Element SWCCB	Addresses all software item SCR/SDRs involving external interface changes and prior build release SU changes as they are generated by the IPT software personnel and approves all requests to postpone qualification test cases to later builds
Element SCM	Manages test database, the source code, SCR/SDRs for software item baseline changes to source code, and test documentation as generated by the software IPTs and software test engineers

**SIQT Process Activities.** Figure 5.9 is an example of a flow chart representation of the SIQT process. As shown in the figure, the SIQT process involves 16 recommended tasks that are organized into four groups covering the preparation, dry run, performance, and the analysis and recording of SIQT results.



SRS = Software Requirements Specification  
 RTVM = Requirements Test Verification Matrix  
 STP = Software Test Plan  
 IFCD = Interface Control Document  
 IDD = Interface Design Description  
 STD = Software Test Description  
 SDD = Software Design Description

STR = Software Test Report  
 SAD = Software Architecture Description  
 SCR = Software Change Request  
 SDR = Software Discrepancy Report  
 IS = Interface Specification  
 SCM = Software Configuration Management  
 SQA = Software Quality Assurance  
 TRR = Test Readiness Review

Figure 5.9. SIQT Process Flow—Example

### 5.9.1 Independence in Software Item Qualification Testing

SIQT **must** be performed to demonstrate that the SI meets the software and interface requirements allocated to that build. **To ensure objectivity, the tests must be performed by independent software test engineers.** They may be either personnel not involved in any of the development activities up to this point or other software developers who have not been involved with the coding and integration activities for the SI being tested.

### 5.9.2 Testing on the Target Computer System

To the maximum extent possible, SIQT should be performed on the target computer system, or as close as possible to the operational target hardware configuration, to demonstrate that there are no hardware or operating system incompatibilities. Operation on the target computer can also provide useful measurements of the computer resource utilization. If the SIQT is to be performed on a compatible system, an analysis **must** be conducted to determine that the computer resource utilization



requirements can be met. All target hardware/computer system(s) used for testing **must** follow Configuration Management Policies and Procedures.

### 5.9.3 Preparing for Software Item Qualification Testing

**Objective.** The objective of preparing for SIQT is to finalize, through reviews or inspections, the STD and the STP. The software developers **must** define and record the test preparations, test cases, and test procedures to be used for SIQT as well as the traceability between the test cases, test procedure steps, and the SI and software interface requirements. In addition to writing the STD, the test engineers **must** run all or portions of the STD, update test scenario procedures and databases, and perform other necessary test activities to prepare for the SIQT Dry Run. **The STP and STD must be baselined and placed under CM control prior to the run for record.**

**Approach.** All requirements in preparation for SIQT, as described in paragraph 5.9.3 of TOR-3537B, **must** be addressed including verification of all software:

- Requirements under conditions as close as possible to those that the software will encounter in the operational environment
- Interface requirements using the actual interfaces wherever possible or high-fidelity simulations
- Specialty engineering requirements such as supportability, testability, reliability, maintainability, availability, safety, security, and human systems integration, as applicable
- Reliability requirements including fault detection, isolation, and recovery
- Stress testing performed including worst-case scenarios

Reference to “testing” during SIQT should not be confused with the “verification method” of test. Software qualification testing may require the use of all verification methods including Inspection, Analysis, Demonstration, and Test. The STD, supported by the STP, **must** provide test case descriptions with test procedures for each test case, special test environments, and test sequencing requirements for all SRS and IRS requirements allocated to the SI build.

Commercial Off-The-Shelf (COTS) software, reuse code, or newly developed software can be used to satisfy and verify software requirements during qualification testing. During SIQT, applicable resource measurements **must** be collected typically including CPU, memory, storage, and bandwidth data.

For some SRS or IRS requirements, it may not be possible, or practical, to fully test the requirement at the SI level for the current SI build. As a result, it may be necessary to satisfy the requirement using unit integration or SU tests rather than a SI test. This situation can result when data associated with a SI requirement is not accessible at the SI level or the test requires an inordinate amount of time or costs to perform. If system hardware or special test environments are not ready for the current build test, the tests can be deferred to segment or system testing.

The STP and STD should be evaluated at a document peer review. Once the review changes are incorporated into the documents, they **must** be baselined and further modifications handled via change control. Supporting test software (simulations/emulations) and data should also be prepared. The updated STP and STD **must** also contain test cases and descriptions for safety-critical requirements from previous builds. For the final build, the documents **must** provide test cases and descriptions for the final build plus regression test cases and descriptions of software requirements from all previous builds. Table 5.9.3 is an example description of the tasks applicable to the SIQT preparation tasks.



Table 5.9.3. SIQT Preparation Tasks—Example

Tasks	Inputs	Subtasks	Outputs
1. Review STP	<ul style="list-style-type: none"> <li>STP</li> </ul>	<ul style="list-style-type: none"> <li>Announce the peer review and disseminate schedules and STP in advance</li> <li>Conduct the peer review and document peer review results</li> <li>Schedule next revision and review dates</li> </ul>	<ul style="list-style-type: none"> <li>Comments against the STP</li> <li>Action Items</li> </ul>
2. Update Baseline STP for SIQT	<ul style="list-style-type: none"> <li>See Figure 5.9 Inputs</li> <li>System and Element/Segment Use Cases</li> </ul>	<ul style="list-style-type: none"> <li>Define test environment and test schedule</li> <li>Develop test categories</li> <li>Identify processes for conducting the SIQT</li> <li>Identify assumptions and constraints</li> <li>Document items in the STP</li> </ul>	<ul style="list-style-type: none"> <li>STP</li> </ul>
3. Approve STP	<ul style="list-style-type: none"> <li>Comments against the STP</li> <li>STP</li> <li>Action Items from previous review</li> <li>Updates to the STP</li> </ul>	<ul style="list-style-type: none"> <li>Update STP based on review comments</li> <li>Provide STP for re-review</li> <li>If re-review is required, conduct, document and schedule next revision and review</li> <li>Obtain the proper approvals</li> <li>Provide to SCM and to Document Control for distribution</li> </ul>	<ul style="list-style-type: none"> <li>Approved STP</li> </ul>
4. Develop Baseline STD	<ul style="list-style-type: none"> <li>See Figure 5.9 Inputs</li> <li>Integration Test Cases</li> <li>SCRs/SDRs</li> </ul>	<ul style="list-style-type: none"> <li>Map SRS and IRS requirements to test cases</li> <li>Populate RTVM with requirement to test-case mapping data</li> <li>Identify requirements to be verified at the SU or SI level</li> <li>Develop automated test scenarios and databases</li> <li>Develop test procedures including post test analysis steps</li> <li>Update scenarios and test databases as required</li> <li>Document anomalies in an SCR</li> </ul>	<ul style="list-style-type: none"> <li>STD</li> <li>Test Scenarios and Databases</li> <li>Updated Requirements Database</li> <li>SCRs/SDRs</li> </ul>
5. Review STD	<ul style="list-style-type: none"> <li>STDs</li> <li>STP</li> <li>SCRs/SDRs</li> </ul>	<ul style="list-style-type: none"> <li>Announce review and disseminate schedules and SIQT STDs</li> <li>Conduct review and document review results</li> <li>Schedule next revision and review dates</li> </ul>	<ul style="list-style-type: none"> <li>Comments against the STD</li> <li>Action Items</li> </ul>
6. Update and Approve STD	<ul style="list-style-type: none"> <li>Comments against the STD</li> <li>SCRs/SDRs</li> <li>Completed Action Items from previous review</li> </ul>	<ul style="list-style-type: none"> <li>Update STD based on review comments</li> <li>Provide STD for re-review</li> <li>If re-review required, conduct and document review</li> <li>Schedule next revision and review if needed</li> <li>If no more review time is required, obtain proper approvals</li> <li>Provide STD to SCM and Document Control for distribution</li> </ul>	<ul style="list-style-type: none"> <li>Approved STD</li> </ul>

#### 5.9.4 Dry Run of Software Item Qualification Testing

**Objectives.** The objectives of the SIQT dry run are to exercise the test cases and test procedures to ensure that they are complete and accurate, and that the SI is ready for witnessed testing. SIQT readiness testing also verifies that all necessary test data and the test environment are under proper SCM control and are adequate for verifying the software requirements. Table 5.9.4 contains an example of the tasks applicable to SIQT dry run.

Table 5.9.4. SIQT Dry Run Tasks—Example

Tasks	Inputs	Subtasks	Outputs
7. Dry-Run STD and Test Procedures	<ul style="list-style-type: none"> <li>Approved STD</li> <li>SCM controlled scenarios and databases</li> <li>SCM controlled software build</li> <li>CM controlled HW test bed and test environment</li> </ul>	<ul style="list-style-type: none"> <li>Ensure all test software and hardware are available and are the correct version</li> <li>Ensure that all approved test software and the software to be tested are under SCM control</li> <li>Execute test and post-test analysis as documented in the STD</li> <li>Redline procedures and obtain SQA approval</li> <li>Update scenarios and databases as needed</li> </ul>	<ul style="list-style-type: none"> <li>Approved STD with SQA Approved Red-lines</li> </ul>
8. Document Dry-Run Test Results	<ul style="list-style-type: none"> <li>STP</li> <li>Approved STD with approved redlines</li> </ul>	<ul style="list-style-type: none"> <li>Document Dry Run results in the test log</li> <li>Document all anomalies in an SDRs</li> </ul>	<ul style="list-style-type: none"> <li>SCRs/SDRs</li> <li>Test Logs</li> </ul>

**Approach.** Testers **must** obtain the appropriate software SI build from the SDL. When all required elements are assembled, the tests, including required regression tests, should proceed per the procedures specified in the STD. The testers **must** collect, analyze and record the outputs, logs, test notes, and results (problems, errors, and discrepancies noted by the tester). No modification to the SI, test data, or environment should be made until after the dry run is complete and the results documented in the SDF. SQA may audit the dry run and all test results.

After the test procedure is executed, and SQA captures the redlines, data from the test execution **must** be analyzed to determine if the software under test produced the correct results and whether the SRS and IRS requirements allocated to the test procedure were actually verified. Results of the post-test analysis may be software changes (documented in SDRs), procedure changes, test data/scenario changes, or test environment changes.

In cases where requirements are not satisfied, SCR/SDRs **must** be generated and the changes handled by the appropriate corrective action process. In cases where test descriptions require modification, the procedures **must** be redlined and approved by SQA. Retesting **must** be required for all modified SUs, test cases, and test descriptions, and any additional SUs and test cases that directly interact with the modified SIs and test cases.

### 5.9.5 Performing Software Item Qualification Testing

**Objective.** The objective of performing SIQT is to formally execute the test procedures as documented in the STD, using products under SCM control, and in a witnessed test environment. The approach in this task should begin with the Test Readiness Review (TRR) that should be described in an SDP appendix covering software reviews. The material presented at the TRR should include: SU testing, SU test results and SI dry-run results; formal test environment description (hardware, test tools, and associated software); formal test approach; SI requirements verification at a lower level; test schedules; and SIQT tasks as described in the SI STP and the SI STD.

**Approach.** The TRR ensures that all necessary test documentation, materials, and personnel are ready, and that coordinated test schedules are in place. The actual execution of the SIQT should be essentially the same as the dry run. The testers usually begin by ensuring that all necessary software test data and tools are available. Testers obtain the appropriate SI test build from the SDL. When all required elements are assembled, the tests, including required regression tests, **must** proceed per the procedures specified in the STD.

The test team **must** collect and record the outputs, logs, test notes, and results. They **must** execute all tests specified for the current build, and they **must** perform regression testing on all safety-critical requirements from previous builds. The primary difference from the dry run testing is that the



performance of the SIQT is normally witnessed by SQA and the CSWE (or designee) and optionally by the customer and the Independent Verification and Validation (IV&V) agent. Reasonable notice of the tests **must** be provided to the customer and the IV&V agent to permit them the opportunity to attend. Table 5.9.5 is an example of the tasks applicable to performing the formal SIQT.

Table 5.9.5. Perform Formal SIQT Tasks—Example

Tasks	Inputs	Subtasks	Outputs
9. Conduct SIQT TRR	<ul style="list-style-type: none"> <li>• TRR entrance and exit criteria</li> <li>• Test environment</li> <li>• Approved STD</li> <li>• SDFs</li> <li>• SRS and IRS</li> <li>• SCM controlled software build, test software &amp; hardware</li> <li>• Open SCR/SDRs</li> <li>• Test Logs</li> </ul>	<ul style="list-style-type: none"> <li>• Review SU test and integration test status</li> <li>• Review SIQT dry-run status and open SCR/SDR status</li> <li>• Review test environment status and STD status</li> <li>• Review test limitations and test schedule</li> <li>• Prepare TRR Test Log</li> <li>• Ensure all test software and hardware are available and are the correct version</li> <li>• Ensure that all test hardware and software are under SCM control</li> <li>• Assess SIQT test readiness based on the above</li> </ul>	<ul style="list-style-type: none"> <li>• Pass/fail status of TRR Exit Criteria</li> <li>• Test Logs</li> </ul>
10. Execute SIQT Test Procedures	<ul style="list-style-type: none"> <li>• Approved SIQT STDs</li> <li>• Test environment</li> <li>• TRR results (test logs)</li> <li>• SCM controlled build</li> <li>• Open SCR/SDRs</li> </ul>	<ul style="list-style-type: none"> <li>• Perform test steps as documented in the STD</li> <li>• Perform analysis steps as documented in the STD</li> <li>• Perform retesting as required</li> </ul>	<ul style="list-style-type: none"> <li>• Completed SIQT Testing</li> <li>• Test Logs</li> </ul>
11. Document SIQT Test Results	<ul style="list-style-type: none"> <li>• Completed SIQT testing</li> </ul>	<ul style="list-style-type: none"> <li>• Prepare the test log</li> <li>• Document anomalies in SDRs and rework source code or STDs to eliminate problems</li> <li>• Document verification status for SRS requirements and obtain SEIT approval</li> <li>• SCM updates to baseline documents</li> </ul>	<ul style="list-style-type: none"> <li>• SCR/SDRs</li> <li>• Test Logs</li> <li>• Revisions to Code or Test Procedures</li> <li>• SQA Audits and Reports</li> </ul>
12. Perform Regression Testing	<ul style="list-style-type: none"> <li>• SI builds</li> <li>• Test Results</li> <li>• SCR/SDRs</li> </ul>	<ul style="list-style-type: none"> <li>• Perform regression testing to accommodate new functions, problems, or changes in the current build</li> </ul>	<ul style="list-style-type: none"> <li>• SCR/SDRs</li> <li>• Test Results</li> </ul>

### 5.9.6 SIQT Revision and Retesting

**Objective.** The objective of the revision activity during SIQT is to rework source code or test descriptions to eliminate any problems identified during qualification testing. Appropriate portions of the SI **must** be retested to verify that the changes have been successful and that other problems have not been produced as side effects. The objective of re-testing is to verify that applicable SCR and SDR fixes are properly implemented and that selected existing functionality is still performing per software and interface requirements after the SCR and SDR fixes have been implemented (Regression Testing).

**Approach.** The test results, and documented problems, **must** be evaluated by software developers to determine if changes need to be made to the SUs or the STD. Regression testing of affected SIQT test cases **must** be performed after any modification to previously tested software. In addition to modifications made to SUs to fix defects, regression testing can also include regression tests of SIQT test procedures from the last build to show that the current build has not broken any software requirements that were previously verified.

All modifications to the source code **must** be handled as SCR/SDRs by the appropriate change control board. Unit-level retesting **must** be required for all modified procedures and functions. Modified SIs require retesting of safety-critical requirements and previously failed test cases.



Products from previous activities **must** be reviewed for possible changes resulting from the implemented software changes, and then updated as appropriate.

This activity **must** be repeated as needed until all test cases have met the test case success criteria. In some cases, uncompleted or failed tests can be postponed until a later build if approved by the SWCCB. Re-testing objectives **must** be reviewed by the appropriate SEIT IPT prior to the testing and preparation for testing. An updated set of the STP or STD should not be mandatory for each iteration of re-testing. **STRs must be provided at the end of SIQT testing.**

### 5.9.7 Analyzing and Recording SIQT Results

**Objective.** The objective of analyzing and recording SIQT results is to finalize the SIQT activity by:

- Documenting test results in the Software Test Report (STR) (for MC-I and SS-I software) or in the SDF (for SS-2 software)
- Performing a review of the STR, or verifying the capture of the test results in the SDF
- Conducting an optional Test Exit Review (TER), that may also be called a Post Test Review (PTR) or Build Turnover Review (BTR)

**Approach.** The results of the SIQT **must** be analyzed for completeness and documented in a STR or captured in the SDF. This documentation should be prepared by software test engineers who performed the SI tests. The completed documentation is subject to a document review.

Software Test Reports **must** be baselined once all review modifications have been incorporated. Any SI documentation, notes, or data that are not incorporated into the STR should be captured in the SDF. For MC-I and SS-I software developed in multiple builds, an STR **must** be prepared and reviewed after each build. **The STR, when approved, must be maintained under CM control.** The intent of recording SIQT test results in the SIQT STRs is to document and finalize the test activity and effectively capture test results. Table 5.9.7 is an example of the tasks applicable to analyzing and recording SIQT results.

Table 5.9.7. Analyzing and Recording SIQT Results—Example

Task	Inputs	Subtasks	Outputs
13. Develop STR	<ul style="list-style-type: none"> <li>• SIQT Test Logs</li> <li>• SCRs/SDRs</li> <li>• "As Run" STD</li> </ul>	<ul style="list-style-type: none"> <li>• Review SIQT Test Logs, As-Run STD and SCR/SDRs</li> <li>• Review verification status of SRS requirements</li> <li>• Document all of the above in STR</li> </ul>	<ul style="list-style-type: none"> <li>• STR</li> <li>• SCRs/SDRs</li> </ul>
14. Analyze Test Results	<ul style="list-style-type: none"> <li>• STR</li> <li>• SCRs/SDRs</li> </ul>	<ul style="list-style-type: none"> <li>• Perform root cause analysis of test anomalies as documented in the SCRs/SDRs</li> <li>• Obtain CCB approval of SCR/SDR resolution plan</li> </ul>	<ul style="list-style-type: none"> <li>• Root Cause Analysis</li> <li>• Resolution Plan</li> </ul>
15. Review STR	<ul style="list-style-type: none"> <li>• SIQT STR</li> <li>• SCRs/SDRs</li> <li>• SIQT Test Logs</li> <li>• "As Run" STD</li> </ul>	<ul style="list-style-type: none"> <li>• Announce review and disseminate schedules and STR</li> <li>• Conduct a TER (PTR or BTR) and document the review results</li> </ul>	<ul style="list-style-type: none"> <li>• Comments against the SIQT STRs</li> <li>• Action Items</li> </ul>
16. Update and Approve STR	<ul style="list-style-type: none"> <li>• Comments against the preliminary SIQT STRs and STR; Action Items from previous reviews; STR Updates</li> </ul>	<ul style="list-style-type: none"> <li>• Update STR based on review comments</li> <li>• If re-review required, conduct and document review</li> <li>• Schedule next revision and review if needed</li> <li>• If no more review time is required, obtain proper approvals</li> <li>• Provide STDs to SCM and Document Control for distribution</li> </ul>	<ul style="list-style-type: none"> <li>• Approved SIQT STR</li> <li>• SQA Audits and Reports</li> </ul>

## 5.10 Software/Hardware Item Integration and Testing

**Introduction.** This subsection of the SDP addresses the objectives, approach, documentation, responsibilities and activities of the Software/Hardware Item (SI/HI) Integration and Testing (I&T) activity. In accordance with TOR-3537B, the SI/HI I&T activity **must** be described in four paragraphs:

- Preparing for SI/HI integration and testing (paragraph 5.10.1)
- Performing SI/HI integration and testing (paragraph 5.10.2)
- Revision and retesting (paragraph 5.10.3 in the TOR but paragraph 5.10.4 in this Guidebook)
- Analyzing and recording SI/HI integration and test results (paragraph 5.10.4 but paragraph 5.10.3 in this Guidebook)

The SI/HI I&T process involves integrating SIs with interfacing SIs and HIs, testing the resulting groupings to determine if they work together as intended, and continuing this process until all SIs and HIs in the system are integrated and tested. Generally, SI/HI I&T is the first integration of the full software system with the target hardware. **SI/HI I&T may test an entire element or segment or a portion thereof.** For activities involving hardware integration, the software team is usually in a support role to SEIT.

**Objectives.** The principal objectives of SI/HI Integration and Testing are to:

- Perform the individual SI-to-SI integrations and SI-to-HI integrations to produce the complete software build for each successive level of test and verify its integration success
- Integrate software into the target hardware system and verify integration success
- Verify SI to SI and SI to HI interface requirements compliance
- Support and successfully complete integration and qualification testing at each level of integration

**Approach.** The Software Master Build Plan (SMBP) should be updated to define the SI functionality that is planned to be operational for each build. The integration sequencing should be documented in the SMBP and the overall approach to I&T documented in the Master Test Plan (MTP)—sometimes called the System Test and Evaluation Plan (STEP). Although integration of software with the hardware is a critical objective of this activity, some aspects of the hardware integration may not be able to be performed until the full system integration.

The following is an example scenario of SI/HI I&T:

1. SIQT for the spacecraft bus software is performed in the flight test bed
2. SIQT for the payload software is performed in the payload test bed
3. The two test beds are connected and the software-to-software interfaces between the spacecraft bus and the payload is tested
4. The spacecraft bus software and the payload software are integrated into the actual vehicle
5. The flight and payload software and hardware are integrated.

In addition, there may be early integration points using the two test beds so that all the spacecraft software and payload software interfaces do not have to wait until the software is completely finished to be integrated and tested.



Table 5.10 is a summary example of the readiness criteria for this activity in terms of entry and exit criteria, verification criteria to ensure completion of the required tasks, and the measurements typically collected during this activity.

Table 5.10. Readiness Criteria: Software/Hardware Item Integration and Testing—Example

Entry Criteria		Exit Criteria	
<ul style="list-style-type: none"><li>• HW/SW integration approach is defined and approved in the Master Test Plan (MTP).</li><li>• IPT software personnel are requested by SEIT to support the HW/SW integration activities.</li><li>• The executable software product has completed the SIQT process and is capable of supporting HW/SW integration.</li><li>• The software release to be integrated and the integration database are under control of the Software Development Library.</li><li>• The integration database is defined and integration tools are available.</li></ul>		<ul style="list-style-type: none"><li>• HW/SW Integration and Testing is successfully completed including an action plan to close remaining SDRs.</li><li>• SW test and SW management reviews and approves the SDR/SCR closure plan.</li><li>• Regression testing is completed and accepted by the SW test lead.</li></ul>	
Verification Criteria			
<ul style="list-style-type: none"><li>• SI-SI and SI-HI integration is verified and accepted by the SW test lead.</li><li>• SQA performs process/product audits for ongoing product engineering activities per SDP subsection 5.16.</li><li>• SW test lead reviews regression test logs and accepts completion of the regression testing.</li></ul>			
Measurements			
<ul style="list-style-type: none"><li>• Test Coverage: Number of requirements tested and passed</li><li>• Number of test cases—planned versus actual</li><li>• Percent of interfaces tested</li><li>• SDRs and SCRs opened and closed. Aging data, origin and root cause analysis.</li></ul>			
			(see subsection 5.20)

The flowchart in Figure 5.10 shows the inputs, outputs, and relationships between the four SI/HI I&T tasks: prepare, perform, analyze/record results, and revision/retest.

**Roles and Responsibilities.** Generally, the SEIT Test and Evaluation team is responsible for performing SI/HI I&T and the software role in this activity consists primarily of support tasks. However, the software test team has a vital role—especially for the SI-to-SI integration tasks. The SI-to-HI integration tasks should involve both hardware and software system engineers/testers. A comprehensive description of the tasks involved in the full SI/HI integration and testing activity is not directly addressed in this subsection—focus is on the support provided by the Software IPT personnel. All testing **must** be run using documented test descriptions developed collaboratively by the software and SEIT test engineers.

**Work Products.** The documentation produced during this activity is focused on testing the integration of software and hardware at various levels of the cumulative integration. During this activity the SI/HI I&T test cases, test procedures, test drivers, test scenarios, test stubs, databases, and other needed test data are produced. The activity concludes with the SI/HI I&T test results and preparation of Software Discrepancy Reports (SDRs) for all problems encountered.

Developed concurrently with the SI/HI I&T activity is an updated Software Test Description (STD as baselined during software qualification testing and discussed in subsection 5.9), a baselined Software Product Specification (SPS as described in paragraph 5.12.1), and the baselined Software Version Descriptions (SVD as described in paragraph 5.12.2) supporting the current software release.

**Approach.** The software developers, and software system engineers, assigned to perform integration of the SIs, **must** develop an integration strategy that defines a systematic approach for integrating the SIs into the complete software release. Issues such as SI-SI interfaces, inter-SI timing and sequencing, and simulations or emulations (for external interfaces) are examples of the issues that go into determining the order of integration of the SIs.

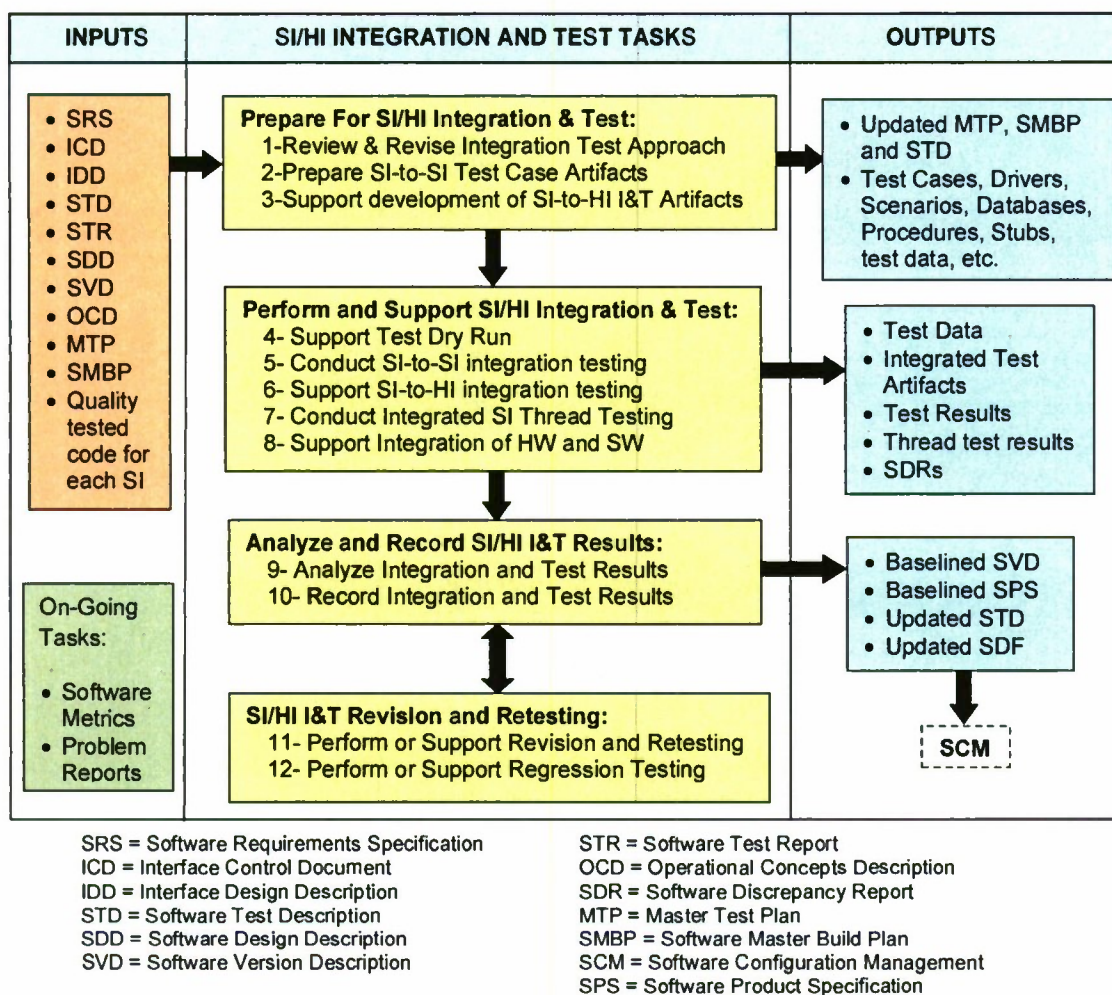


Figure 5.10. Hardware/Software Item Integration and Test Process—Example

### 5.10.1 Preparing For SI/Hi Integration and Testing

**Objectives.** The objectives of preparing for SI-SI integration and SI-HI integration testing are to finalize the MTP, SMBP and STD and develop integration test artifacts (test cases, test procedures, test drivers, test scenarios, test stubs, databases and test data) necessary to verify the success of the integration effort. Table 5.10.1 contains examples of the tasks applicable to preparation for the SI/Hi I&T activity.



Table 5.10.1. SI/Hi Integration and Testing Preparation Tasks—Example

Tasks	Inputs	Subtasks	Outputs
1. Review and Revise Integration Test Approach	<ul style="list-style-type: none"> <li>See Inputs to Figure 5.10</li> </ul>	<ul style="list-style-type: none"> <li>Review SI/Hi I&amp;T approach in the MTP, the test sequence in the SMBP, test requirements in the STD, and update if necessary</li> <li>Define functional capability threads</li> <li>Develop integrated schedule and activity/dependency network</li> <li>Identify assumptions/constraints</li> <li>Document planning results</li> <li>Obtain proper approvals</li> </ul>	<ul style="list-style-type: none"> <li>Updated MTP, SMBP, and STD</li> <li>Updated SI/Hi integration plans</li> <li>Updated integration schedule</li> <li>Action items</li> </ul>
2. Prepare SI-to-SI Test Case Artifacts	<ul style="list-style-type: none"> <li>Outputs and Action Items from Task 1</li> </ul>	<ul style="list-style-type: none"> <li>Prepare SI-to-SI Test Cases, Test Procedures, Test Drivers, Test Scenarios, Test Stubs, Databases, and other test data as needed</li> <li>Update MTP, SMBP, and STD</li> <li>Define integration test, threads, and test cases</li> </ul>	<ul style="list-style-type: none"> <li>Test Cases, Test Procedures, Test Drivers, Test Scenarios, Test Stubs, Databases, etc.</li> <li>Functional Capability Thread Descriptions</li> </ul>
3. Support Development of SI-to-Hi I&T Artifacts	<ul style="list-style-type: none"> <li>Outputs from Tasks 1 and 2</li> </ul>	<ul style="list-style-type: none"> <li>Support development of SI-to-Hi Test Cases, Test Procedures, Test Drivers, Test Scenarios, Test Stubs, Databases, and needed test data</li> <li>Update MTP and SMBP, if necessary</li> </ul>	<ul style="list-style-type: none"> <li>SI-to-Hi Test Cases, Test Procedures, Test Drivers, Test Scenarios, Test Stubs, Databases, and test data as applicable.</li> </ul>

Hardware and software system engineers **must** collaborate in the preparation of appropriate test description information for the hardware/software integration that needs to be accomplished during this activity. They review in-progress SI/Hi I&T Software Test Descriptions, provide recommendations to test engineers (including software test equipment needed), and ensure that test cases and corresponding test procedures are sufficiently defined in the updated STD to verify the success of each partial integration. An updated STD for SI/Hi I&T should contain at a minimum:

- The overall test description and test environment for SI-to-SI and SI-to-Hi integration and testing.
- Specific test cases and corresponding test procedures to verify correct execution of SI-to-SI and SI-to-Hi interfaces including:
  - End-to-end functional capabilities
  - Sequencing and timing of events and data flows
  - All requirements allocated to software
  - Stress testing including worst-case scenarios
  - Start-up, termination, and restart procedures
  - Fault detection, isolation, and recovery handling
  - Performance testing including input/output data rates, timing, and accuracy requirements
  - Operation of multiple SIs on a single computer platform, where applicable
  - Integrated error and exception handling capabilities
  - Limit and boundary conditions
  - Resource utilization measurements (e.g., CPU, memory, storage, and bandwidth)
- Input data definitions (e.g., data files, databases, etc.)
- Required simulations and emulations needed for external or hardware interfaces
- Specific output data to be collected and recorded in the appropriate SDF
- The expected results and success criteria

The SI/Hi I&T **must** be performed using the target hardware in a configuration that is as close as possible to the operational configuration. All reuse software, including legacy reuse and COTS

software, **must** also undergo the SI/HI I&T process. The software IPTs provide software system engineers and test engineers along with applicable software test support items, expertise, and training as required in using the software.

### 5.10.2 Performing SI/HI Integration and Testing

**Objective.** The objective of performing SI/HI I&T is to integrate and test software in accordance with integration and test strategies in the approved MTP and SMBP. The five principal tasks are: Support the Dry Run; Conduct SI-to-SI Integration Testing; Support SI-to-HI Integration Testing; Conduct Integrated SI Thread Testing; and Support Integration of HW/SW. Integration testing may also be called Element Qualification Test or Factory Acceptance Test. Table 5.10.2 contains examples of the tasks applicable to performing the SI/HI I&T activity.

Table 5.10.2. Performing SI/HI Integration and Testing Tasks—Example

Tasks	Inputs	Subtasks	Outputs
4. Support Test Dry Run	<ul style="list-style-type: none"> <li>Approved test Integration procedures, plans and schedules</li> <li>SRS for the build</li> <li>Integrated SW-HW build</li> <li>Integrated SI thread test cases</li> <li>Integration test data and tools</li> <li>Hardware test equipment</li> </ul>	<ul style="list-style-type: none"> <li>Ensure all test SW and HW are available, the correct version and under CM control</li> <li>Perform test cases and procedures</li> <li>Document HW-SW dry run integration test results in test log</li> <li>Generate SDRs/SCRs as applicable</li> <li>Redline procedures and obtain SQA approval</li> </ul>	<ul style="list-style-type: none"> <li>Approved integration procedures</li> <li>Integrated test stubs, drivers, and scenarios</li> <li>Test results in log</li> <li>Integrated build</li> </ul>
5. Conduct SI-to-SI Integration Testing	<ul style="list-style-type: none"> <li>Approved STP</li> <li>SW plans and schedules</li> <li>SRS for the build</li> <li>SIs from SCM</li> <li>Integration test data and tools</li> </ul>	<ul style="list-style-type: none"> <li>Integrate SIs in accordance with integration test procedures in the MTP and SMBP</li> <li>Develop SI thread test cases in accordance with the SRS and MTP</li> </ul>	<ul style="list-style-type: none"> <li>Integrated builds</li> <li>Integrated SI thread test cases</li> <li>SCRs/SDRs</li> </ul>
6. Support SI-to-HI Integration Testing	<ul style="list-style-type: none"> <li>Same inputs as Activity 5 plus:               <ul style="list-style-type: none"> <li>Integrated HW-SW test drivers, scenarios, and stubs</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Same tasks as Activity 5 plus:               <ul style="list-style-type: none"> <li>Record test logs</li> <li>Document status for interface requirements</li> <li>SQA audit and review test status</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Integrated HW/SW builds</li> <li>EAT completion</li> <li>SCRs/SDRs</li> <li>Code and test procedure revisions</li> <li>SQA audit report</li> </ul>
7. Conduct Integrated SI Thread Testing	<ul style="list-style-type: none"> <li>Approved test integration procedures</li> <li>SW plans and schedules</li> <li>SRS for Build</li> <li>Integrated SIs</li> <li>Integrated SI thread test cases</li> <li>Integration test data and test tools</li> </ul>	<ul style="list-style-type: none"> <li>Develop SI thread test procedures, stubs, drivers and scenarios in accordance with integration plan and SI test cases</li> <li>Develop HW/SW integration test cases and procedures</li> <li>Perform SI thread test procedures</li> <li>Document SI thread test results</li> <li>Generate SDRs/SDRs, if applicable</li> </ul>	<ul style="list-style-type: none"> <li>Integrated build thread test procedures, drivers, stubs, and scenarios</li> <li>Thread test results</li> <li>Integrated and tested builds</li> <li>Test cases</li> </ul>
8. Support Integration of HW and SW	<ul style="list-style-type: none"> <li>Approved test Integration procedures, plans and schedules</li> <li>Integrated and tested SIs</li> <li>Target hardware</li> <li>HW/SW Integration test cases</li> <li>SRS for Build &amp; SDRs/SCRs</li> <li>Integration test data and tools</li> </ul>	<ul style="list-style-type: none"> <li>Obtain Integrated and tested builds from SCM</li> <li>Integrate software build with the target hardware.</li> <li>Rework source code if required in response to approved SDRs and SCR.</li> </ul>	<ul style="list-style-type: none"> <li>HW/SW Integrated build</li> <li>HW/SW integration test cases</li> <li>SDRs/SCRs</li> </ul>

**Approach.** Software integrators normally begin the integration of SIs by ensuring that all SIs to be integrated and all necessary data and tools are available and ready. Software test engineers support CCB and SWCBB corrective actions on any soft-related errors, request re-execution of build procedures as required, and accept SI builds upon satisfactory verification. When all required elements are assembled, integration proceeds.



The software test engineers **must** run test cases using the test procedures, as specified in the STD. They collect or record the outputs, logs, test notes, and results. All problems, errors, and discrepancies **must** be noted. Similarly, segment test engineers run the hardware/software integration tests as defined in the test descriptions, collect and record test results and problems.

### 5.10.3 Analyzing and Recording SI/HI Integration and Test Results

**Objectives.** The objectives of analyzing and recording SI/HI integration and test results are to: (a) analyze integration tests results to ensure the tests have been successfully completed; and (b) to document the respective test data and results as required. Table 5.10.3 contains examples of the tasks applicable to analyzing and recording of the SI/HI I&T activity.

**Approach.** After all SIs and HIs have been successfully integrated and tested, the integration and test team **must** review the test results for consistency and completeness and to verify that the integration test data and results have been documented. If discrepancies or problems are found, then the portion of the integration in question **must** be retested. It is also a good idea for an independent reviewer, not involved with the segment hardware/software integration testing or a SEIT team member, to perform an independent review, however, independent reviews are not required by the TOR-3537B standard.

Table 5.10.3. Analyzing and Recording SI/HI Integration and Test Tasks—Example

Tasks	Inputs	Subtasks	Outputs
9. Analyze Integration and Test Results	<ul style="list-style-type: none"> <li>• DRs</li> <li>• Approved Test Integration Procedures</li> <li>• Integration Test Results</li> </ul>	<ul style="list-style-type: none"> <li>• Collect test results</li> <li>• Analyze test data to ensure proper processing of input data by each procedure and correct output data</li> </ul>	<ul style="list-style-type: none"> <li>• Analysis results</li> </ul>
10. Record Integration and Test Results	<ul style="list-style-type: none"> <li>• Integration and Test Results</li> <li>• Analysis Results</li> </ul>	<ul style="list-style-type: none"> <li>• Collect test and analysis results</li> <li>• Ensure that results are correctly and completely recorded</li> </ul>	<ul style="list-style-type: none"> <li>• Build Integration Release Notice</li> <li>• Released build for site and system testing</li> <li>• Document test and analysis results</li> </ul>

Once the independent reviewer signifies that the integration and testing is complete, and the integration testing was successfully completed, the release is baselined. At the last stage of integration and testing, the test results are normally documented by SCM in a Build Integration Release Notice and the build is then ready for system testing.

### 5.10.4 SI/HI I&T Revision and Retesting

**Objectives.** The objectives of retesting are to verify that changes and applicable SDR/SCR modifications have been implemented correctly and that the functionality is performing in accordance with requirements after the fixes have been completed. Changes can also involve test procedures, test data, etc. as well as code changes. Re-integration and retesting **must** then be performed to verify that the changes have been successful and have not caused side effects. The documented problems **must** be evaluated by software developers to determine the necessary changes to SIs, SUs, or to the test descriptions. Table 5.10.4 contains examples of the tasks applicable to revision and testing for the SI/HI I&T activity.

Table 5.10.4. Revision and Retesting SI/Hi Integration and Test Tasks—Example

Tasks	Inputs	Subtasks	Outputs
11. Perform or Support Revisions and Retesting	<ul style="list-style-type: none"> <li>Same inputs as Activity 5 or 6</li> </ul>	<ul style="list-style-type: none"> <li>Same tasks as Activity 5 or 6 plus:               <ul style="list-style-type: none"> <li>Perform DR fixes</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Same inputs as Activity 5 or 6</li> </ul>
12. Perform or Support Regression Testing	<ul style="list-style-type: none"> <li>Same inputs as Activity 5 or 6</li> </ul>	<ul style="list-style-type: none"> <li>Same tasks as Activity 5 or 6 plus:               <ul style="list-style-type: none"> <li>Perform DR fixes</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Same inputs as Activity 5 or 6</li> </ul>

**Approach.** Retesting is performed to show that a problem is fixed and the test case executes properly. Regression testing is performed to show that the fix did not break anything that was previously tested and working properly before the fix. In cases where software requires changes, SDRs or SCRs are generated and the changes are handled by the Corrective Action Process (CAP). In cases where test descriptions require modification, the changes, identified in the SDRs or SCRs, **must** be made by software test engineers and a version history included in the test description to record the changes made.

Modified software requires retesting for the integration tests that previously failed and for any tests that are dependent on the failed tests. Similarly, test description changes require retesting of the changed tests plus tests that are dependent on the results of the changes.

This process **must** be repeated until all the SIs and HIs have been successfully integrated and all tests have been completed. If the element IPT lead determines it is impractical to complete certain changes until a later build, then SCRs/SDRs **must** be used to document and control the modifications and integration testing that still needs to be performed. The CCB at the element level **must** approve all such delays.

## 5.11 System Qualification Testing

**Introduction.** This subsection of the SDP is focused on the objectives, approach, work products, roles, and responsibilities of System Qualification Testing (SQT). The SQT activity involves verifying that the system requirements have been met—including the system interface requirements.

Subsection 5.11 is also applicable to the verification of requirements at all levels above verification of the software requirements. Those levels typically include subsystems, elements, segments, and the system. The major qualification tasks at each level are similar but details of the required tests, procedures and documentation may be different. If a system is developed in multiple builds, qualification testing of the full system will not occur until the final build.

There are seven paragraphs prescribed by TOR-3537B for SQT/SAT. The last five of these tasks (paragraphs 5.11.3 through 5.11.7) are the sequential processing steps of the SQT activity. The System Qualification Testing activity **must** be described in the following paragraphs in the SDP:

- Independence in System Qualification Testing (paragraph 5.11.1)
- Testing on the target computer system (paragraph 5.11.2)
- Preparing for System Qualification Testing (paragraph 5.11.3)
- Dry run of System Qualification Testing (paragraph 5.11.4)
- Performing System Qualification Testing (paragraph 5.11.5)
- Revision and retesting (paragraph 5.11.6 in the TOR but paragraph 5.11.7 in this Guidebook)
- Analyzing and recording System Qualification Test Results (paragraph 5.11.7 but paragraph 5.11.6 in this Guidebook)



**SQT Objectives.** System Qualification Testing is the formal test demonstrating that the system software functional and interface requirements have been met for that release of the system. At the system level, SQT is focused on testing the integrated hardware/software system against the system requirements. The Technical Requirements Document (TRD) and Interface Specifications (IS) define the system requirements, and the Software Master Build Plan (SMBP) defines what SI functionality is to be operational for each release and what segment releases are used for each system release. This activity **must** fully test the integrated software with the system hardware it interfaces with. This activity also tests those portions of the hardware/software integration that have been previously completed.

**SQT Approach.** The SQT activity consists of the following similar tasks:

- Prepare the SQT software test data
- Perform SQT test readiness on the target computer hardware to ensure that the tests in the STDs are complete and accurate
  - Perform formal SAT/SQT: Conduct a SQT Test Readiness Review (TRR)
  - Execute the tests using the SQT test procedures and record the test results, problems, and anomalies
- Analyze test results and document the test data and results
- Record test results in the SDF

Table 5.11 is an example summary of the readiness criteria in terms of entry and exit criteria, verification criteria to ensure completion of the required tasks, and the required measurements to be collected during the SQT activity.

Table 5.11. Readiness Criteria: System/Segment Qualification Testing—Example

Entry Criteria	Exit Criteria
<ul style="list-style-type: none"> <li>• System/Segment test plan and approach is defined and approved.</li> <li>• The software IPT is requested by SEIT to support SQT activities.</li> <li>• The executable software product is capable of supporting SQT.</li> <li>• The software release to be integrated and the integration database are under control of the Master Software Development Library (MSDL).</li> <li>• The System test database requirements have been defined.</li> </ul>	<ul style="list-style-type: none"> <li>• The System Test Readiness Review (TRR) is successfully completed.</li> <li>• Verification of test cases and procedures (e.g., Peer Reviews) have been completed.</li> <li>• The release being tested is ready and accepted.</li> <li>• Required test databases are created, populated, and accepted by the test conductor.</li> <li>• System/Segment testing is successfully completed with an action plan generated to close remaining DRs/CRs.</li> <li>• Regression testing is completed and accepted.</li> <li>• Software and system management reviews and approves the DR closure plan.</li> </ul>
Verification Criteria	
<ul style="list-style-type: none"> <li>• Releases provided by the SDL to the MSDL are verified and accepted by the test conductor.</li> <li>• SQA performs process/product audits for ongoing product engineering tasks per SDP subsection 5.16.</li> <li>• Test conductor reviews the test database for completeness.</li> <li>• Test conductor and software Test Lead reviews regression test logs and accepts completion of the regression testing.</li> </ul>	
Measurements	
<ul style="list-style-type: none"> <li>• Test Coverage: Number of requirements tested and passed</li> <li>• Percent of paths tested</li> <li>• SDRs, DRs, SCRs, and CRs opened and closed</li> </ul>	
(see subsection 5.20)	

Discrepancy Reports and Change Requests (DR/CR) usually replace SDR/CDRs at the system level of testing and software problems are allocated to the Software IPT. If DR/CRs fixes are incorporated into the software under test, constituting a new sub-release, then portions of the SQT test procedures are re-run to verify that applicable fixes are implemented and working correctly. In addition, it **must**

be determined that selected pre-existing functionality is still performing per software and interface requirements after the fixes have been implemented. See paragraph 5.11.6 for details on performing revision and re-testing.

**Roles and Responsibilities.** Depending on where in the Specification Tree hierarchy the testing is performed, SQT is the responsibility of the system, segment, subsystem or element integration and test team. Software developers and software test engineers have no formal role in System Qualification Testing but typically provide support as needed. Test description preparation, test execution, and test results documentation are performed by the system, segment, subsystem or element test engineers.

For SQT, software developers implement software changes resulting from DR/CRs generated during this activity and support the system test engineers in these activities. Also, software engineers may support the System Functional Configuration Audit (System FCA) and the System Physical Configuration Audit (System PCA), if required, as outlined in the Master Test Plan (MTP), and discussed in paragraph 5.14.4. The MTP is sometimes called the System or Integrated Test and Evaluation Plan (STEP or ITEP).

**Work Products.** SQT must be performed using documented test descriptions developed by the test engineers. The Software Version Description (SVD) and Software Product Specification (SPS) are updated concurrently if required. Additional products may be specified in a development site's SDP Annex.

#### 5.11.1 Independence in System Qualification Testing

System qualification testing demonstrates that the system, segment, subsystem or element meets the performance and interface requirements allocated to it for each release. System qualification testing is normally the responsibility of the program-level SEIT, however, at the lower levels, SQT can be performed by the segment, subsystem or element test engineers. To ensure objectivity, the tests must be performed by independent test engineers. System test engineers have no role in the software development process and so are inherently independent testers of the software. In any case, software engineers support the SQT process.

#### 5.11.2 Testing On the Target Computer System

System Qualification Testing must be performed on the target hardware system, in the operational configuration, to the maximum extent possible to demonstrate that there are no hardware/software incompatibilities. Testing on the target hardware verifies a successful hardware/software integration and interoperability. Operation on the target computer or target comparable systems also enables the collection and analysis of measurements of the computer resource utilization.

#### 5.11.3 Preparing For System Qualification Testing

**Objectives.** The objective of preparing for System qualification testing is to prepare and finalize, through reviews or inspections, the segment test description and data. Once the review changes are incorporated into the documents, they should be baselined and submitted to documentation control. In addition, all supporting test software (simulations/emulations) and data must be prepared. System test plans, procedures, and test data should be prepared at the appropriate level of testing and is often prepared by the SEIT at the level of the test.

**Approach.** Separate test descriptions should be generated for each release. They should contain test cases and procedures for the requirements of the current release, plus those safety-critical requirements from previous releases. For the final release, the documents provide test cases and



descriptions for the final release software requirements plus regression test cases and descriptions for the software requirements from previous releases.

Software IPTs should review in-progress test plans and test descriptions, and provide recommendations to system, segment, subsystem or element test engineers. The software IPT personnel also assist in determining needed software test data, equipment, and test support items as well as providing expertise and training in using the software. The test data and the software **must** be placed under CM control prior to testing. The IPT software personnel can also support applicable readiness reviews.

#### 5.11.4 Dry Run of System Qualification Testing

**Objectives.** The objective of the System qualification testing dry run is to exercise the test cases and test procedures to ensure that they are complete and accurate, and that the segment is ready for witnessed testing. The test engineers normally begin by ensuring that all necessary data and tools are available. They **must** support SWCCB corrective actions on any release errors, request re-execution of release procedures as required, and accept software releases upon satisfactory verification.

**Approach.** When required elements are assembled, testers execute the procedures and collect or record the outputs, logs, test notes, and results. They execute all tests specified for each release, and perform regression testing on all safety-critical requirements from previous releases. All problems, errors, and discrepancies **must** be noted by the tester. No modification to the SIs, hardware, configuration, test data, or environment should be made until after the dry run is complete and the results are documented.

There is no formal software developer role for system dry-run testing, except to assist test engineers in analyzing test discrepancies and generating DRs/CRs. If software code requires changes, SDRs/SCRs **must** be generated. In cases where test procedures require modification, the procedures **must** be redlined and approved by SQA. Retesting is required for all modified software, test cases, test descriptions, and test cases that directly interact with the modified software and test cases.

#### 5.11.5 Performing System Qualification Testing

**Objectives.** The objective of performing System qualification testing is to execute the test procedures in a formal and witnessed test environment using products under CM control. This task normally begins with the Test Readiness Review (TRR). This review ensures that all necessary test documentation, equipment, materials, and personnel are ready, and that coordinated test schedules are in place.

**Approach.** The actual execution of the SQT is the same as the dry run, except that the performance of the testing **must** be witnessed by SQA and optionally by the Program Office and/or its representatives. Reasonable notice of the tests **must** be provided to permit the Program Office an opportunity to attend.

There are no formal software developer requirements for this system test, except for analyzing software discrepancies, generating CRs/DRs as needed, and implementing needed software code changes resulting from the CRs/DRs.

#### 5.11.6 Analyzing and Recording System Qualification Test Results

**Objectives.** The objectives of analyzing and recording SQT test results are to: (a) analyze SQT tests results to ensure the tests have been successfully completed; and (b) document test results in the SDF and in the Software Version Description (SVD) if required.

**Approach.** There are no formal software developer roles in system qualification testing other than supporting the SIQT at the level being tested. However, results of the qualification tests **must** be analyzed for completeness and then recorded in the SDF by the test engineers who performed the tests. For software developed in multiple releases, test results **must** be prepared, reviewed, and recorded after each release unless a program decision has been made to defer the higher level (system) test until all the software releases are complete. Segment SVDs should be updated after each release.

#### 5.11.7 System Qualification Testing Revision and Retesting

**Objectives.** The objective of the revision and retesting activity is to rework the source code or test descriptions to eliminate problems identified during the qualification testing, and then to retest the appropriate portions of the system to verify that the changes have been successful and have not produced side effects. The test results and documented problems should be evaluated by software developers to determine the necessary changes to the software and test descriptions.

**Approach.** Unit-level retesting is required for all modified procedures and functions. Modified software releases require retesting of all safety-critical requirements and previously failed test cases. There are no formal software developer roles for this task, except for implementing software code changes resulting from the change control process. All modifications to the source code **must** be handled as DRs/CRs.

Revision and retesting **must** be repeated as needed until all test cases have met the test case success criteria. In some cases, resolving incomplete or failed tests can be postponed until a later release if:

- a. no segment external interface is involved
- b. specific functionality is not required by another SI for the release
- c. the delay is approved by the Change Control Board (CCB)

#### 5.12 Preparing for Software Transition to Operations

This activity is concerned with the preparation, installation, and checkout of the executable software, on the target system, at a customer or user site. Upon successful completion of the System Qualification Test (SQT) for the final build, and closure of all DRs/CRs allocated to software, SDRs and SCRs that can be closed, the software development cycle is completed and the software is ready for transfer to the customer for government system testing. It may also be necessary to provide interim releases to development sites if needed to facilitate their development and testing process.

Prior to actually releasing the software for use, there remains software and documentation preparation work that must be completed. This subsection of the SDP addresses the tasks necessary to prepare the software and software-related products necessary for a user to run the software. In accordance with TOR-3537B, the Preparing for Software Transition to Operations activity **must** be described by four paragraphs in the SDP:

- Preparing the executable software (paragraph 5.12.1)
- Preparing version descriptions for user sites (paragraph 5.12.2)
- Preparing User Manuals (paragraph 5.12.3)
- Installation at user sites (paragraph 5.12.4)

**Objectives.** Preparing software for use ensures that there is a smooth transition of software into the actual operational system. These tasks **must** begin well before completion of the SQT. This activity



cannot be fully completed until SQT of the final build has been completed and all DRs/CRs allocated to software plus SCRs/SDRs that can be resolved have been dispositioned.

**Approach.** Although the focus of these tasks, to ensure a smooth transition, is at the end of the development lifecycle, consideration of these tasks should occur concurrently with design, development, and testing throughout the lifecycle. During each design period, new or updated user and operations manuals can be prepared for review by the customer and users. Draft versions of the Software Transition Plan (STrP) should be started during the software design activity.

For final deliveries, the tasks and products of this activity **must** be in compliance with the Master Test Plan (MTP). This planning **must** be coordinated with the hardware installation schedules. Schedules are established and resources and personnel required for installation and support are identified. This activity also involves the planning, preparation, and presentation of required user training.

Software installation and checkout tasks are performed by software test personnel at the user site. When SQT has been completed, SCM prepares the software product(s) for use in accordance with the CM Plan. For software in the Ground Segment, the products are stored on media formatted as required for installation at the operational site. For on-board software, the preparation of the executable software includes downloading it into the actual flight hardware. Similarly, software for user equipment is usually downloaded into target processors.

### 5.12.1 Preparing the Executable Software

This activity includes preparation of the specific executable code and source files for each SI, batch files, COTS, command, data, or other software files needed to install and operate the software on the target computer(s). This data is packaged in the Software Product Specification (SPS) as described in paragraph 5.13.4. The list of data to be prepared should be specified in the SPS executable software paragraph.

### 5.12.2 Preparing Version Descriptions for User Sites

Each software release requires a Software Version Description (SVD) document. Format and contents of the SVD are described in the Data Item Description (DID) listed in Appendix D and the completed SVD requires a document review prior to release. The SVD is primarily composed of lists that should include, as applicable:

- Complete identification of all material released including numbers, dates, abbreviations, version and release numbers, physical media, and documentation
- Inventory of all computer files that make up the software being released
- History of all changes incorporated since the previous version
- Site-unique data contained in the new version
- Related documentation not included in current release
- Installation instructions
- Possible problems and known errors

### 5.12.3 Preparing User Manuals

Software customer user manuals are required to be prepared for ground segment software, and software in user equipment with a human interface. However, not all of the user manuals need to be produced by all programs because the full set of user manuals normally has some duplication. On-

board software does not require user manuals nor does equipment with embedded software. The customer and the developer **must** determine which user manuals are appropriate for each system. User manuals or user guides should be produced for SS-1 and SS-2 software. Existing vendor documentation can be used for COTS/Reuse software.

There are various types of user manuals as described below. For each of the required user manual types, a separate document should be written for each segment. Segments can optionally write multiple user manuals covering one or more SIs, rather than a single user manual for the entire segment. This approach is recommended in cases where different users run selected SIs within the segment. All of the documents below should follow the DID product descriptions, listed in Appendix D, and they require a document review prior to release.

#### 5.12.3.1 Software User Manuals

A Software User Manual (SUM) **must** be written to provide information needed at the customer site if required by the program. Its detailed format and contents are described in the SUM DID listed in Appendix D. The SUM describes, in depth, how to use the software and includes, as applicable:

- An inventory of software required to be installed for the software to operate
- Resources needed for a user to install and run the software
- Software overview including logical components, performance characteristics, etc.
- Procedures to access the software for first time or occasional users
- Detailed procedures for using the software including organization, capabilities, conventions used, backup, recovery and messages (Note: The detailed procedures may be organized by “operator position” rather than following the software structure)

#### 5.12.3.2 Computer Operation Manuals

Computer Operation Manuals (COM) are written to provide information needed by the customer site to operate the target computers. A COM is typically needed only if the hardware is unique or new. Its detailed contents are described in the COM DID listed in Appendix D. The COM describes the computer system operations data including, as applicable:

- Computer system preparation, power on/off, initiation, and shutdown
- Operating procedures including input/output, monitoring, and off-line procedures
- Diagnostic features, procedures, and tools

#### 5.12.3.3 Other User/Operator Software Product Descriptions (Optional)

There are two optional user/operator manuals: the Software Input/Output Manual (SIOM); and the Software Center Operations Manual (SCOM). The SIOM and SCOM are used for software systems installed in a computer center or other centralized or networked software installation. There is some overlap in these documents, as well as with other user manuals, so the appropriate set must be determined for each system.

**SIOM.** Detailed contents of the SIOM are described in the SIOM DID listed in Appendix D. The SIOM describes how to prepare input to, and interpret output from, the software including, as applicable:



- An inventory of software files and databases needed to access the software
- Resources needed to access the software
- Organization and operation of the software from a users point of view
- Contingencies, security, and problem reporting procedures
- Input conditions, formats, rules, vocabulary, and examples of each type of input
- Output descriptions, formats, vocabulary use, examples, and error diagnostics
- Query procedures including file formats, capabilities, and instructions
- Terminal processing procedures covering capabilities, displays, updates, retrieval, error correction, and termination

**SCOM.** Detailed contents of the SCOM are described in the SCOM DID listed in Appendix D. The SCOM describes required installation procedures including:

- An inventory of software required to be installed for the software to operate
- Resources needed for a user to install and operate the software
- Software overview including logical components, performance characteristics, etc.
- Detailed description of runs to be performed including: run inventory, phasing, diagnostic procedures, error messages, control inputs, input and output files and reports, and procedures for restart and recovery

#### **5.12.4 Installation at User Sites**

Preparation for installation of the system at customer sites should be handled by the development organization. The developers should be responsible for the system setup and checkout, development of user training, provision of user training, and initial user assistance.

The Software Installation Plan (SIP) is a plan for installing software at user sites. It includes preparations, user training, and conversion from existing systems. It is prepared only when the developer is involved in the installation of software at user sites, and when the installation process is sufficiently complex to warrant the need for a SIP. If the software is embedded in a hardware-software system, the installation plan for the system usually includes a System Installation Plan covering both hardware and software. Detailed contents of the SIP are described in the SIP DID listed in Appendix D.

The installation of the system at user sites for government requirements verification testing and the final delivery after requirements verification is handled as specified in the contract. Software developers and SCM support this process by providing technical support and implementing changes that result from the government testing prior to final delivery.

#### **5.13 Preparing For Software Transition to Maintenance**

This subsection addresses the software and documentation preparation work that must be completed to transition the application and support software for the performance of system maintenance. In accordance with TOR-3537B, the Preparing for Software Transition to Maintenance activity **must** be described by nine paragraphs in the SDP:

- Preparing the executable software (paragraph 5.13.1)
- Preparing source files (paragraph 5.13.2)
- Preparing the version descriptions for the maintenance site (paragraph 5.13.3)

- Preparing the “as built” Software Item design and related information (paragraph 5.13.4)
- Updating the system/subsystem design description (paragraph 5.13.5)
- Updating the software requirements (paragraph 5.13.6)
- Updating the system requirements (paragraph 5.13.7)
- Preparing maintenance manuals (paragraph 5.13.8)
- Transition to the designated maintenance site (paragraph 5.13.9)

If the software is developed in multiple builds, the developer’s planning should identify what software builds are to be transitioned to the maintenance organization. Preparing for software transition in each build is interpreted to include those activities necessary to carry out the transition plans for that build.

**Objectives.** Software transition involves considerable advance planning and preparation that **must** start early in the lifecycle to ensure a smooth transition to the maintenance organization. The tasks and products of this activity **must** be in compliance with the Software Transition Plan (STrP) that defines the plans for transitioning the software, test beds and tools to the maintenance center’s facilities (see paragraph 5.1.5).

**Approach.** The Preparation for Software Transition includes preparation of the documentation and software products required by maintenance personnel at the maintenance center to perform their maintenance tasks. This includes the:

- Executable code for each SI
- Release build files and documentation
- Software Version Descriptions (SVD) for the executable code
- Source Code for each SI
- Applicable test beds and tools
- Contents of the Master Software Development Library (MSDL)

Software maintenance may be performed by the software development organization or by another organization such as: another organization within the company that developed the software; a different development contractor; or by a government maintenance organization.

The preparation of maintenance manuals, such as the Computer Programming Manual (CPM) and Firmware Support Manual (FSM), should begin early in the software design activity and continue into subsequent activities as pertinent information becomes available.

The final updated versions of the system and software requirements and design descriptions are also included as needed. Finally, the preparation may include the planning, preparation, and presentation of maintenance training as required by the contract.

### 5.13.1 Preparing the Executable Software

The executable software is prepared using the SVD (see paragraph 5.12.2) for each SI or related collection of SIs, and includes the executable code, batch, command, data, test, support, or other software files needed to install and operate the application and support software on the target computers. The results **must** include all applicable items in the executable software section of the Software Product Specification (SPS). Format and contents of the SPS are described in the Data Item Description (DID) listed in Appendix D.



### 5.13.2 Preparing Source Files

The final versions of all source code files should be assembled from the Master Software Development Library (MSDL—see paragraph 5.2.3) and transferred to the desired transfer media per the agreement with the selected maintenance center in accordance with the Software Configuration Management Plan (see subsection 5.14). The results **must** include all applicable items in the source file section of the SPS.

### 5.13.3 Preparing Version Descriptions for the Maintenance Site

As discussed in paragraph 5.12.2, a Software Version Description (SVD) **must** be prepared for each software delivery to the maintenance site. The SVD provides an inventory of the software contents for the final build. It also provides a history of version changes, unique-to-site data, related documentation, installation instructions, and possible problems for each SI within the segment. The maintenance center's version requires the additional version information related to the maintenance center's tools, SDFs, test software, and documentation.

### 5.13.4 Preparing the “As Built” Software Item Design and Related Information

Software item designs **must** be documented in SDDs and IDD for MC-I and SS-I software. In addition, software development work products for MC-I and SS-I software are documented in SDFs. These documents and products **must** be updated with each build to maintain them as the “as built” software throughout the development process. They are provided to the maintenance center to describe the SI design. The SPS provides additional information needed by the maintenance center to maintain the application and support software. The SDP **must** define and record:

- The methods to be used to verify copies of the software
- The measured computer hardware resource utilization for the SIs
- Other information as needed to maintain the software
- Traceability between the software item's source files and Software Units
- Traceability between the computer hardware resource utilization measurements and the SI requirements concerning them

The SDP **must** indicate that the results of this task are placed in the qualification, software maintenance, and traceability sections of the SPS. A document review prior to release is required. The SPS is the “as-built” version and includes or references, as applicable:

- Executable software and source files for the SI
- The “as-built” versions of the SDD, IDD, and DBDD
- Compilation, build, and modification procedures
- Measured utilization of computer hardware resources
- Requirements traceability data
- Packaging requirements

### 5.13.5 Updating the System/Subsystem Design Description

The system design **must** be documented in the System/Subsystem Design Description (SSDD). Once baselined, the SSDD **must** be maintained under configuration control throughout the development process. Software development work products evolve from the system requirements definition activity from the software requirements and design activities to the “as built” system.

If, throughout the development process, these products are maintained as the “as built” system, there is no special updating required at the end of the development process—except for modifications that result from finalizing SCRs/SDRs after the last system build qualification test.

### 5.13.6 Updating the Software Requirements

The baselined SRSs and IRSs **must** be maintained under configuration control throughout the software lifecycle process and should require no special updating in preparation for software transitioning. If this is not the case, the SRS and IRS **must** be updated to contain the current set of approved requirements that the software transition to maintenance is to meet.

### 5.13.7 Updating the System Requirements

The system specifications should be controlled throughout the system lifecycle process, and should require no specific updating in preparation for software transitioning. If this is not the case, the System/Subsystem Specifications (SSS), and system level IRSs, **must** be updated.

### 5.13.8 Preparing Maintenance Manuals

Preparation of a Software Maintenance Plan (SMP), early in the development process, is discussed in paragraph 5.26.3. A comprehensive SMP is a major asset in assuring the timely availability of adequate facilities, support software, personnel, and documentation so that the software can be maintained in an operational and sustainable condition. Also, if contractually required, the segments **must** prepare Computer Programming Manuals (CPM) and Firmware Support Manuals (FSM) for the maintenance center. For each of the required maintenance manuals, a separate document should be written for each segment. In addition, a Software Center Operators Manual (SCOM), produced as described in subparagraph 5.12.3.3, may be useful to the maintenance center.

Segments can optionally write multiple documents for each maintenance manual type covering individual computers or firmware devices, rather than single maintenance manuals for the entire segment. This approach is recommended when computers and firmware devices are maintained in separate locations.

#### 5.13.8.1 Computer Programming Manual

The Computer Programming Manual (CPM) provides information needed by the maintenance center to program the computers on which the application and support software run. Contents and format of the CPM are located in the DID number identified in Appendix D. The CPM requires a document review prior to release to the MSDL. **The CPM is primarily intended for unique or newly developed computers.** The CPM basically describes the programming environment and includes, as applicable:

- The components and configuration of the computer system
- Equipment needed including operating characteristics, capabilities, and limitations
- Programming features, control instructions, subroutines, interrupt procedures, timing, memory protection, etc.
- Description of instructions including use, syntax, execution time, etc.
- Input and output control programming instructions
- Special programming techniques including error detection and diagnostic features



### 5.13.8.2 Firmware Support Manual

A Firmware Support Manual (FSM) provides the information needed by the maintenance center to program and reprogram firmware devices with application or support software. Contents and format of the FSM are located in the DID number identified in Appendix D. The FSM requires a document review prior to release to the MSDL. The FSM describes the details of a programmed firmware device and includes, as applicable:

- Relevant vendor information plus model number and a complete physical description
- Operational and environmental limitations
- Software to be programmed into the device and equipment and software needed
- Procedures for programming, reprogramming, installation, and repair

### 5.13.9 Transition to the Designated Maintenance Site

The completed software, support environment, and the above mentioned documentation **must** be delivered to the maintenance center facility in accordance with the Software Transition Plan (STrP). Contents and format of the STrP are located in the DID number identified in Appendix D. The STrP identifies and describes all resources needed to maintain the deliverable software. It includes, as applicable, the following resources needed to maintain the deliverable software:

- The facilities, hardware, software, documentation, personnel, and other resources
- Interrelationships of components and recommended procedures
- Training plans, anticipated changes, and transition plans

The maintenance center staff should be responsible for installation and check out of the system application and support software in the maintenance center and demonstrating that the software can be regenerated in the maintenance center. The software IPT supports these tasks including the transition of licenses, providing training, and other software support to the maintenance center.

## 5.14 Software Configuration Management

Software Configuration Management (SCM) is an essential activity that begins during requirements definition. Formal software control starts with the establishment of the Allocated Baseline, which identifies the SIs that **must** be formally managed in coordination with the Configuration Control Boards (CCB). A software baseline is a specific version of controlled software source code, documentation or data for an increment, build, or release. A requirements baseline involves an approved Software Requirements Specification (SRS) under SCM control. **SCM is responsible for the tasks necessary to control baselined software products and to maintain the status of the baselines throughout the development lifecycle.** The basic SCM responsibilities are to:

- Establish baselines of identified products
  - Identify software configuration items, components, and related products that will be placed under configuration management and when that takes place
  - Establish and maintain a configuration management and change management system for controlling software products
  - Create and release baselines for internal use and for delivery to the customer
- Track and control changes to products
  - Track change requests for the configuration items and control the changes

- Establish and maintain integrity of baselines
  - Establish and maintain records describing configuration items and perform configuration audits to maintain integrity of the configuration baselines

The SCM activities are performed by both the contractor team as well as the Acquisition Program Office. A general description of the division of SCM responsibilities is shown in Table 5.14-1.

Table 5.14-1. Division of SCM Responsibilities—Example

Contractor Team	Acquisition Program Office
Defines and documents software configuration management processes	Reviews software processes for quality and ensures process compliance
Implements software configuration management tools and environments	Reviews software configuration management tools and environments for quality and process compliance
Conducts software configuration management boards	Participates in software configuration management boards
Ensures the integrity of all software configuration items	Audits delivered software products for baseline integrity
	Incentivizes the contractor to control software baselines through award fee

Details of the SCM activity **must** be covered either in subsection 5.14 of the SDP, in an addendum to the SDP, or in a Software Configuration Management Plan (SCMP). A good option is to include an SCM overview in the SDP with the details documented in the SCMP. The SCMP is described in paragraph 5.14.2.

**Configuration Management Control Levels.** Typically, SCM has a three-tiered configuration management scheme, as shown in Figure 5.14, consisting of program-level, segment-level, and software development site controls. SCM tasks should be performed at each software development site using the site's Software Development Library (SDL) for configuration control of the developed products. On large programs, the SDLs may be subdivided into two levels—at the developer level and at the element or IPT level. The SDLs move up the levels to reach the Master Software Development Library (MSDL).

SCM establishes and maintains the integrity of specified software work products, controls and promotes stable baselines, maintains status accounting of the baselines throughout the life of a project, and controls the build process through product delivery. Responsibility for SCM should reside with the Software Group Lead. The principal SCM performers within segment and development sites are the SCM Lead and the SCM Librarian.

**Library Levels.** An example of electronic SDL partitioning was shown in Figure 5.2.3. The specific organization of the SDL and MSDL **must** be tailored for applicability to each program, however, Table 5.14-2 is an example of the library levels, names, and who controls them. Specific guidance regarding structure and control of the software libraries should be provided in the SDP and/or the SCMP.



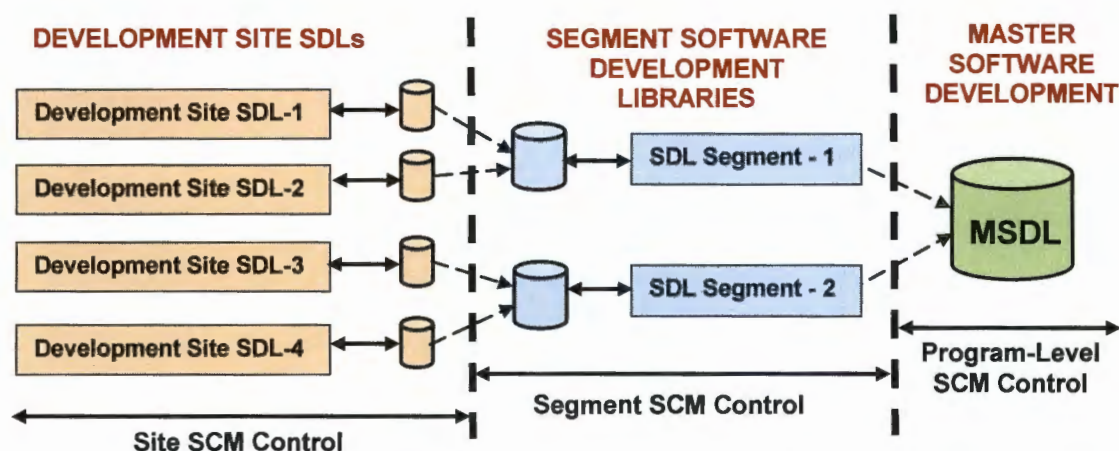


Figure 5.14. Relationship of the SDLs to the MSDL—Example

Table 5.14-2. Software Library Levels and Controls—Example

Library Level	Library Name	Controlled By	SCCB Controls	CM Controls
1	Developer	Software Developer	SCCB Controls Promotion Between Levels	CM Controls Access to Each Level
2	Integration and Test	Site SCM		
3	Software Build and Qualification Test	SCM at Site or Segment		
4	Segment Qualification Test	Segment SCM		
5	System Qualification Test	System SCM		

In accordance with TOR-3537B, the Software Configuration Management activity **must** be described by five paragraphs in the SDP:

- Configuration Identification (paragraph 5.14.1)
- Configuration Control (paragraph 5.14.2)
- Configuration Status Accounting (paragraph 5.14.3)
- Configuration Audits (paragraph 5.14.4)
- Packaging, Storage, Handling and Delivery (paragraph 5.14.5)

#### 5.14.1 Configuration Identification

Configuration Identification consists of identifying the software products to be placed under configuration control. These software products are the development products identified in subsections 5.3 through 5.11 of the SDP, the hardware and software in the software development environment, plus other documents and data as required by the contract.

The output of Configuration Identification is the configuration controlled list of configuration items. Each software product **must** be uniquely identified by the software IPT with Program Unique Identifiers (PUI). The Chief Software Engineer (CSWE), or designee, should be responsible for ensuring that a common, and unique, software PUI scheme is used across the entire program. The identification scheme **must** be at the level at which the software entities will be controlled, for example, computer files, electronic media, documents, SIs, SUs, and hardware elements.

CASE tools used to support Configuration Identification should be identified as well as how their features (e.g., versioning, branching, and labeling) are used to track and control promotion and delivery of deliverable items.

### 5.14.2 Configuration Control

Configuration control is the systematic control of modifications to baseline products throughout the product's lifecycle. The SDP or the SCMP **must** describe the SCM process for controlling baselined products and establishing common SCM change procedures. Segment or element SCM procedures may be provided in the segment or element SDP Annexes. The policies and process for approving and implementing changes to baselined software products **must** be defined in the SDP or SCMP. More detailed operational procedures may augment the overall direction provided in the SCMP.

#### 5.14.2.1 Software Configuration Management Plan

The SCMP documents the policies and procedures for conducting required software configuration management for all SIs. The SCMP establishes the plan for creating and maintaining a uniform system of configuration identification, control, status accounting, and audit for the software work products throughout the software development process.

**Organization of the SCMP.** The SCMP can be organized into five sections:

- **Section 1:** "Introduction" presents and defines the scope and purpose of the SCMP
- **Section 2:** "Applicable Documents" lists the compliance document(s) and other documents that are referenced, or related to, the SCMP
- **Section 3:** "Organization and Resources" describes the overall structure of the Software CM Organization, personnel, and resources to be employed
- **Section 4:** "Software Configuration Management Activities" details of the major Software CM functions and activities covered at a higher level in SDP paragraphs 5.14.1 through 5.14.5
- **Section 5:** "Glossary" lists the abbreviations and acronyms

**Objectives of the SCMP.** The objective of the SCMP is to define the process to be used by SCM personnel in managing the configuration control of the software work products. Specifically, the SCMP should provide the following guidelines and direction:

- Identifies the software development baseline identification
- Provides change control and visibility of the changes to software work products through configuration control procedures
- Controls for incorporation of all approved software changes, and related documentation, to the Master Software Development Library (MSDL) or the Software Development Library (SDL), and the subsequent release of the SI to Integration and Test, and System Test
- Provides status accounting of software work product changes submitted to the SDL or MSDL
- Ensures that only approved changes are incorporated into the baselined software work products
- Maintains a configuration audit system to ensure that records, which are provided to the MSDL or SDL, are consistent with documentation and software work product identification

Three levels of configuration control are depicted in Figure 5.14. Regardless of the number of CM control levels, overall responsibilities at each level should be defined in the SDP or SCMP including roles and procedures. In addition, the approach to CM related tasks should also be addressed (such as



support to multiple baselines, a distributed development environment, data integrity and data restoration).

#### 5.14.2.2 Configuration Control Boards

On large programs, there is typically a hierarchy of configuration control boards with different levels of control and responsibilities. Software CM supports all of the change boards depicted in Figure 5.14.2.2. At the very top of the hierarchy is the Acquisition Program Office CCB (APO CCB) who participates in, and monitors the activities of, the lower level CCBs as needed.

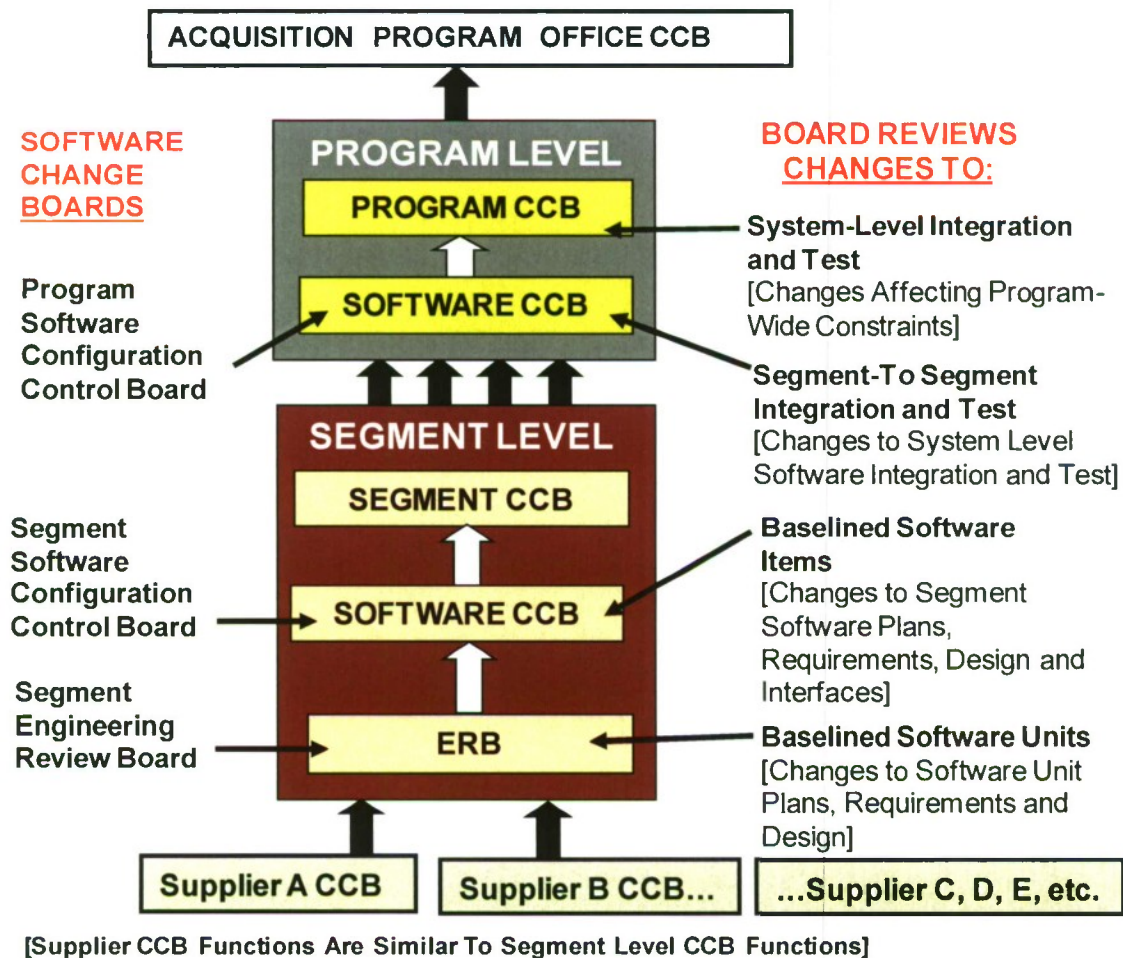


Figure 5.14.2.2. Relationship of the Configuration Control Boards--Example

The program level includes the Configuration Control Board (Program CCB) and the Software Configuration Control Board (SW/CCB). At the segment (or element) level there may also be two lower level boards: the Segment Configuration Control Board (Segment CCB) and the Engineering Review Board (ERB). The suppliers (subcontractors) may also have similar levels of control boards.

The Responsible Software Engineer (RSE), usually the Software Lead, provides support to the segment and element boards. All of these boards **must** be described in the SDP or the SCMP; their functions and relationships are briefly described below:

- **Program Configuration Control Board.** The Program CCB operates under authority of the APO CCB and approves changes to baselined documents that affect cost, schedule, program constraints or scope issues. There may also be a high-level Program Configuration Evaluation Board (CEB). The CCB **must** have cognizance over the program's Allocated and Product Baselines.
- **Software Configuration Control Board.** The SW/CCB operates under authority of the Program CCB; it has control over software changes at the system and segment-to-segment software integration and test level. The SW/CCB **must** have cognizance over the Product Development Baseline for software and the Master Software Development Library (MSDL).
- **Segment Change Control Board.** The Segment CCB **must** have control over changes found in baselined SIs. It reviews SCRs/SDRs, provides impact assessments, assigns appropriate personnel, and oversees the resolution and verification. The Segment CCB **must** have cognizance over the segment's Allocated and Product Development Baselines as well as the segment's Software Development Library (SDL). This function can take place at the element or segment level.
- **Engineering Review Board.** This lower level board performs the same type of functions as the Segment CCB but controls changes found in baselined SUs. This function can take place at the element or segment level.

#### 5.14.3 Configuration Status Accounting

SCM **must** prepare and maintain records of the configuration status for all baselined software products including the maintenance of the records required to support configuration auditing.

Configuration status accounting data includes the current version of each baselined product, a record of changes to the software product since being placed under configuration control, and the recording and reporting of:

- The time at which each baseline was created and when each SI completed the initial build placing the software or database under CM control
- Descriptive information about each SI
- Description and status of each DR (approved, disapproved, awaiting action, incorporated, closed)
- Change status and traceability of changes to controlled software products
- The status of the technical and administrative documentation associated with each product baseline and/or update to a product baseline
- Closure and archive status

Software Version Description (SVD) documents **must** also be prepared for each release to provide a history of version changes, segment/element data, references to related documentation, and references to known problems.

#### 5.14.4 Configuration Audits

SCM **must** perform periodic configuration audits to verify that changes were made in accordance with the corrective action process as described in the SDP or SCMP. SQA can witness and support these audits. Configuration Audits should be used to ensure that submitted software is accompanied by appropriate documentation and approvals, is correctly delivered and merged, and is correctly included in the software builds. The audits **must** ensure that each software entity incorporates only the approved changes scheduled for inclusion at the time of the audit. The degree of formality of the



configuration audits may differ at the different levels of configuration control. The following is an example of text that may be used to augment this paragraph:

**Example Text:**

In the <tool> environment, the SDL and MSDL will be audited at least quarterly. A sampling of software releases is checked against the SVD for correctness and completeness. A check will also be made to ensure that there is a SVD for each release. In the <tool> environment, SCM checks a sampling of closed SDRs/SCRs in the database to ensure they map to the closed SCR/SDRs in the SVD.

**Functional and Physical Configuration Audits.** Software engineers may be requested to support Functional Configuration Audits (FCA), Physical Configuration Audits (PCA), and in some cases the System Verification Review (SVR). Both software and system FCAs and PCAs may be conducted. The SVR is often conducted concurrently with the System FCA.

A Software FCA may be conducted as part of the System Qualification Test or following the SQT. The purpose of a Software FCA is to demonstrate that each SI was successfully tested and complies with the software and interface requirements of its functional requirements and design documentation. To complete the Software FCA, software and system engineers **must** reach a technical understanding on the validity and degree of completeness of the Software Test Reports and the applicable software user documentation. Software FCAs should be conducted on every SI in the system.

The Software FCA is a prerequisite to the Software PCA. The purpose of a Software PCA is to conduct a formal examination of the as-built, and as-coded SI, against its design documentation to establish the product baseline. The Software PCA includes a review of the Software Product Specification (SPS), Interface Design Description (IDD), the Software Version Description (SVD), and all operational and support documentation. Differences between the physical configuration of the SI, and the configuration used for the Software FCA, **must** be identified at the Software PCA. Approved and outstanding changes against the SI **must** also be provided along with approved deviations and waivers to the requirements specifications.

FCAs and PCAs may be conducted on a single SI, a related group of SIs, or incrementally such as blocks. Results of the Software PCA becomes an entrance criteria for the System PCA. The purpose of the System FCAs and PCAs are similar to their software counterparts but they address all of the configuration items covering the entire system.

#### 5.14.5 Packaging, Storage, Handling, and Delivery

The SCM procedures for packaging, storage, handling, and delivery of deliverable software products **must** be provided in the SCMP for both the SDLs and the MSDL. Master copies of delivered software products **must** be maintained in the MSDL for the duration of the contract.

**Packaging.** The package for a software delivery normally consists of the SVD and the CD (media) set and is supported by a verification report. The IPT responsible for the delivery prepares the SVD in collaboration with SCM. The SVD typically requires Engineering Review Board (ERB) and CCB approvals. SCM is responsible for providing the appropriate identification labels. SCM and SQA should perform the package content verification review, using a Verification Checklist. A hardcopy listing of the files should be attached to the hardcopy signed checklist. Upon successful completion of the review, a formal contracts letter should be generated and submitted. Copies of the completed verification checklist and contracts letter **must** be maintained in the CM Library.

**Storage.** The storage requirement can be satisfied through the implementation of multiple support library systems. There should be three basic components of the library system:

1. **Software Library Management System.** The Software Library Management System allows the software to be maintained in a central location, yet each host or client has access. A CASE tool can track the baseline changes, marking the transition throughout the activities of the software lifecycle. SCM must control the software libraries to provide a disciplined structure for development, integration, test, and implementation of software within a controlled, well-defined environment.
2. **Documentation Library Management System.** The Documentation Library Management System is a document repository for the most current approved and controlled documentation.
3. **Data Storage Backup.** The Data Storage Backup component provides daily incremental backups and system backups (performed at least weekly) to ensure recovery from an uncontrollable situation.

**Handling.** A SCM CASE tool is ideally suited to administratively manage the handling of software through the version database directory structure. All elements in the database **must be Read-only—at all times**. They only become Read/Write when they are checked out for updating by an authorized user.

**Delivery.** Release packages for each increment **must** be delivered to the Program SCM organization. All deliveries **must** receive IPT approval prior to shipping. Software deliveries are always on removable media. Installation and checkout of the delivered products at customer-designated facilities should be performed if applicable.

If problems arise during the installation, checkout, or test, these problems **must** be documented on a Software Discrepancy Report (SDR) and resolved (see subsection 5.17).

**Delivery Preparation.** SCM is responsible for accumulating the SIs for milestone deliveries. This responsibility includes overseeing the scheduling, storage, handling, and delivery of the project media. All SIs to be delivered **must** be examined by SQA for specification compliance, SOW compliance, open items to be resolved, DR closure, and test verification status. No delivery should leave the facility without proper authorization from the Program Director or designate. CM should retain records of all deliveries

**Software Version Description.** In preparation for delivery of the system to the customer, the CM organization **must** create a formal SVD document. It contains detailed information on all software components, including COTS and reuse software, and their associated version numbers, plus special instructions required for system installation.

## 5.15 Software Peer Reviews and Product Evaluations

The performance of Software Peer Reviews and Software Product Evaluations are mandatory. In accordance with TOR-3537B, the Software Peer Reviews and Product Evaluations activity **must** be described by the following paragraphs in the SDP:

- Software Peer Reviews (paragraph 5.15.1)
  - Prepare for Software Peer Reviews (subparagraph 5.15.1.1)
  - Conduct Peer Reviews (subparagraph 5.15.1.2)
  - Analyze Peer Review Data (subparagraph 5.15.1.3)



- Software Product Evaluations (paragraph 5.15.2)
  - In-Process and Final Software Product Evaluations (subparagraph 5.15.2.1)
  - Software Product Evaluation Records (subparagraph 5.15.2.2)
  - Independence in Software Product Evaluations (subparagraph 5.15.2.3)

Software products **must** be reviewed for compliance with contract requirements and defined quality evaluation. Peer Reviews are used to perform segment/element software product evaluations on software products as defined in the SDP. The Software Lead is responsible for deciding when to hold software product evaluations. The planning for peer reviews and product evaluations is part of the software development planning process (see paragraph 5.1.1).

The evaluation criteria for each software product **must** be supplied to the Peer Review participants. Figure 5.15 is an example overview of the software Peer Review process.

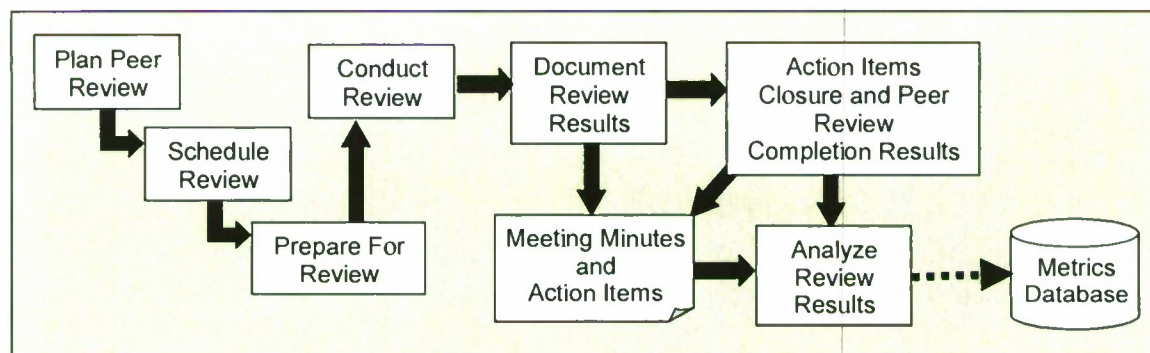


Figure 5.15. Software Peer Review Process Overview—Example

### 5.15.1 Software Peer Reviews

A critical element to the development of the high quality software work products is the performance of software peer reviews. They **must** be an integral part of the development process and focus on the identification and removal of defects as early and efficiently as possible. Peer Reviews evaluate deliverable (and mission critical non-deliverable) work products for correctness, completeness, accuracy, and consistency. They ensure completeness prior to transition from one activity to the next.

Peer reviews also identify areas of needed change and improvement in the product, assure compliance to standards, and ensure satisfaction of functional, performance, and interface requirements. Action items, defects, technical decisions, and measurements resulting from these reviews are documented.

The SDP **must** define the processes to be followed for each type of peer review to be used on each of the software work products and for each software category (described in paragraph 1.2.3). All software work products mission-critical deliverable and non-deliverable software **must** undergo the most formal, robust peer review process for their initial development and for significant changes. Minor changes may undergo a less formal peer review.

The SDP **must** define the peer review processes for preparation before the peer reviews, conducting the peer reviews, and analyzing the resulting peer review data.

### 5.15.1.1 Prepare for Software Peer Reviews

A software project typically adopts the standard peer review process of its parent organization for conducting software peer reviews. Peer reviews **must** be scheduled, planned, and tracked by the Segment and/or Software IPT leads. The software IPT Leads **must**:

- Determine what type of peer review will be conducted on each work product
- Identify key reviewers who **must** participate in each peer review
- Ensure that each software work product satisfies the peer review entry criteria
- Ensure all reviewers understand their roles in the peer review
- Confirm that the participants have reviewed each work product prior to the peer review, using the predetermined review check list for that product.

The segment or element Software Peer Review (SPR) Plan defines the procedures, data collection and reporting for peer review and product evaluations. The implementation of the SPR Plan is the responsibility of the segment/element IPT software personnel Lead. The SPR Plan typically defines the types of Peer Reviews to be held. Peer reviews have varying levels of formality, ranging from Formal Inspections to Colleague Reviews:

- **Formal Inspection.** Formal inspections are performed to verify that software products conform to established technical decisions and applicable standards and procedures. Formal Inspections are the most thorough Peer Review and are conducted initially when the product has reached enough maturity and completeness for a thorough review and when extensive changes are involved. The Software Lead should review all the changes and determine if additional peer review need to be held.
- **Colleague Reviews.** The least formal type of review used primarily to review portions of a mission critical work product during its development to improve the quality of the product during its development. A Colleague Review can be conducted by one person and it may be used for relatively minor updates to software products that have already undergone a formal inspection. Colleague Reviews are not a substitute for required inspections.

### 5.15.1.2 Conduct Peer Reviews

Peer Reviews **must** be conducted, during development tasks, prior to the work products being released to subsequent activities. This subparagraph of the SDP should include a table similar to example Table 5.15.1.2 showing the set of development work products that require peer reviews. A formal, robust type of peer review is used for products establishing deliverable baselines in any development activity. The goal of each Peer Review is to identify work product defects as early as possible in the lifecycle. Defects identified early almost always result in a lower cost to resolve.

**Peer Review Roles:** Peer Reviews should consist of entry/exit criteria, multiple steps, and roles with appropriate participation. The roles normally include moderator, coordinator, recorder, reviewer, monitor and author. Inspection training or skills from past experience should be required for all participants. The Peer Review moderator is instrumental in setting up the inspection, and working with the author to ensure the key reviewers for the work product are present and prepared for the inspection.



Table 5.15.1.2. Software Development Peer Reviews—Example

Development Activity	Work Products Reviewed
System Requirements Analysis (SDP subsection 5.3)	<ul style="list-style-type: none"> <li>• Product specifications</li> <li>• Segment SEIT test plan</li> </ul>
System Architecture Design (SDP subsection 5.4)	<ul style="list-style-type: none"> <li>• Architecture and data models</li> <li>• Interface descriptions</li> <li>• Trade study results</li> </ul>
Software Requirements Analysis (SDP subsection 5.5)	<ul style="list-style-type: none"> <li>• Software requirements documents and data models</li> </ul>
Software Design (SDP subsection 5.6)	<ul style="list-style-type: none"> <li>• Software design documents and data models</li> <li>• Software interface descriptions</li> </ul>
Code And Unit Test (SDP subsection 5.7)	<ul style="list-style-type: none"> <li>• Source files</li> <li>• Unit test cases and procedures</li> </ul>
Integration Testing (SDP subsection 5.8)	<ul style="list-style-type: none"> <li>• Test cases and procedures</li> </ul>
Qualification Testing (SDP subsection 5.9)	<ul style="list-style-type: none"> <li>• Software Test Plan</li> <li>• Software Test Description (cases and procedures)</li> </ul>

Note: The Requirements Test Verification Matrix (RTVM) should be inspected in all activities. The RTVM may be called a Verification Cross Reference Matrix (VCRM).

Project personnel should be trained to perform various inspection roles. These trained inspectors participate to identify defects, improvements, and issues in the product(s) being examined and to contribute in the determination of the product's ability to proceed to the next development activity. Monitors **must** ensure peer reviews are held in accordance with the approved tailored peer review process. Peer review records should be audited to assure they comply with process requirements.

A Coordinator should be selected to ensure the peer review process is followed, to compile and analyze peer review metrics, maintain the list of qualified inspectors, and oversee the implementation of continuous process improvement within the peer review process.

**The Peer Review Package:** A peer review announcement and review package should be distributed at least one week prior to the Peer Review. Participants **must** review the work product against higher-level requirements, using checklists aimed at finding major system problems and for compliance with templates, standards, and guidelines. Findings **must be documented and assigned to responsible individuals as action items for resolution by a due date that is tracked to closure**. Exit Criteria consists of the peer review meeting being conducted on the original work product or extensive rework tasks, work product being updated, and actions being closed.

### 5.15.1.3 Analyze Peer Review Data

The metrics collected during peer reviews can be used to provide visibility into the quality of the produced documents and code and to indicate the need for corrective or process changes. At the end of each development activity, participating organizations analyze the root causes of deficiencies to identify process improvements that can be implemented to avoid future occurrence of those deficiencies.

**Defect Data.** The number of defects expected from a Peer Review depends on when in the lifecycle the review is conducted. Historical data can be used to set expectations for the Peer Reviews conducted on requirements, design, coding, and test products. For software, code count and complexity can be used, as well as experience from prior programs, to predict the number of defects from each lifecycle activity. A defect prevention team can compare predicted versus actual defects to assess the quality of the product at the end of each activity. They can analyze the defect causes to initiate defect prevention process improvements for subsequent builds. Required defect data and associated metrics for peer reviews should be described in the Software Measurement Plan.

**Other Peer Review Data.** Typical peer review data that should be collected at each review and analyzed for potential improvement of the peer review process include:

- Work product(s) under review
- Meeting date, time, location, and completion date
- Number of attendees and preparation time spent by each reviewer
- Size of the work product(s) reviewed
- Inspection type and role assignments
- Time to close action items
- Peer Review Defect List
- Amount of time spent by author in rework of the software work product

### 5.15.2 Software Product Evaluations

As pointed out in TOR-3537B, it is not the intention of SDP paragraph 5.15.1 or 5.15.2 to require separate processes for Software Peer Reviews and Product Evaluations. A developer may choose to have two separate processes; however, it is entirely permissible to accomplish Software Product Evaluations via the Software Peer Review process—as long as the specific requirements for Software Product Evaluations are satisfied. The SDP **must** define the process to be used for Software Product Evaluations.

For SIs developed in multiple builds, software products of each build should be evaluated in the context of the objectives established for that build. As long as the software product meets those objectives it can be considered acceptable even if it is missing functionality designated for inclusion in a later build. If there are SDP Annexes they **must** describe procedures and recording mechanisms to be used at each development site. This information is used to develop schedules for all software product evaluations.

#### 5.15.2.1 In-Process and Final Software Product Evaluations

The TOR-3537B standard requires the software developers to perform in-process evaluations of the software work products. In addition, the standard requires the developer to perform a final evaluation of each product before delivery. Appendix D, in TOR-3537B, contains a list of the minimum in-process software products to be evaluated, along with the minimum evaluation criteria to be used and definitions of the criteria.

The developer should enhance the minimum evaluation criteria in Appendix D to ensure a robust set of evaluation criteria is used that is appropriate to the characteristics of the program and category of software. The enhanced set of qualification criteria for each software product **must** be documented in the SDP. Software product evaluations, both in-process and final, **must** always be performed against a documented set of evaluation criteria.

**Document Reviews.** Document Reviews are conducted on completed software documents, listed in the Contract Data Requirements List (CDRL), for content and accuracy prior to baselining. Document reviews are the final software product evaluations performed by the factory/segment/element IPT software personnel or the software development organization. These reviews should be conducted prior to formal release of new or modified documents to Configuration Data Management (CDM) or Software Configuration Management (SCM). Document reviews **must** be conducted on all CDRL items.



**Risk Analysis.** If a software work product fails to meet its completion readiness criteria, an analysis can be performed to determine the risk of proceeding to the next process. If the risk is acceptable, and appropriate actions and detailed resolution plans have been put in place, then subsequent process tasks may begin. However, advancing insufficient products should be discouraged and limited to cases of extenuating circumstances (e.g., the defect does not impact the implementation process until a latter part of the schedule allowing ample time for repair). The assessment of product readiness and the decision to proceed (including signatures of agreement from key leadership personnel) **must** be documented and retained in the evaluation records (see subparagraph 5.15.2.2).

**Quality Checking.** The key gate to in-process quality checking is not allowing products that do not meet readiness criteria to pass to the next implementation process. This can be achieved by holding a final Engineering Review Board (ERB) review at the end of each implementation process or activity. Product readiness can be assessed by determining if output products are complete, have met their completion criteria, and product defects have been resolved.

#### 5.15.2.2 Software Product Evaluation Records

Software product evaluation records **must be maintained for the duration of the contract**. The accepted product evaluation comments are usually stored in the product's SDF and a summary of findings should be placed in a database for metrics analysis. Closure of product evaluation comments **must** be verified. The following software product evaluation records should be maintained:

- Product evaluation package, including a copy of the product under evaluation
- Review meeting time, date, and attendees
- Review meeting minutes, including technical decisions made, action items captured, and evaluation problems found in the product
- Action item resolutions and closure
- Software work product problem resolutions and closure

Action items from software product evaluations must be tracked to closure. If the software product under evaluation is under a level of configuration management above the individual author/developer, then the problems **must** be document as Problem/Change Reports and handled by the corrective action process (see subsection 5.17). If the product is under control of the individual author/developer, the problems are documented in the SDF and the segment and/or IPT Lead is responsible for ensuring they are properly closed.

#### 5.15.2.3 Independence in Software Product Evaluation

It is absolutely imperative that evaluators of a software work product under evaluation **must not be the persons responsible for developing the software product**. However, this does not preclude software developers from taking part in software product evaluations or reviews of documents they produced. Document reviews should be coordinated by the segment/element IPT software personnel and may include members outside the developer's segment.

### 5.16 Software Quality Assurance

Software Quality Assurance (SQA) performs the planned and systematic pattern of actions necessary to assure that software, and software-related products, satisfy system requirements and support mission success. The SQA organization has a responsibility to provide program management with visibility into the software development process and products by performing independent audits and assessments.

These assessments provide assurance that products and processes conform to contractual requirements and established plans, standards, and procedures. SQA is a member of the Software Engineering Process Group (SEPG) and participates in process improvement tasks. In accordance with TOR-3537B, the Software Quality Assurance activity **must** be described by four paragraphs in the SDP:

- Software Quality Assurance Evaluations (paragraph 5.16.1)
- Software Quality Assurance Records (paragraph 5.16.2)
- Independence in Software Quality Assurance (paragraph 5.16.3)
- Software Quality Assurance Noncompliance Issues (paragraph 5.16.4)

The Software Quality Engineer (SQE) works directly with the development teams to resolve identified problems at the lowest level before elevating them for resolution. The SQE identifies SQA tasks and responsibilities to be implemented on the program.

The evaluations to be performed **must** be identified and the evaluation criteria for those evaluations should be defined and described in a Software Quality Program Plan (SQPP). This plan is normally prepared by SQA to direct the SQE, and the SQE team, in performing the evaluations. SQE tools, techniques, and methodologies to be employed by each software development team member should be defined in the SQPP and can be augmented by segments in their SDP Annexes. The SQPP is maintained and updated as required and is usually an addendum to the SDP. However, in addition to the SQPP, the SDP Data Item Description (DID) requires SQA planning to be addressed in the SDP.

#### 5.16.1 Software Quality Assurance Evaluations

The segment and program SQA organizations both perform two major categories of evaluations: process audits and product reviews. **Process audits** are conducted by SQA to assure effective implementation of the software development process as defined in the SDP. **Product reviews** are performed by program and segment/element level SQAs as a participant in formal reviews. SQA product reviews verify that the software products conform to product requirements. The following example text may be used.

***Example Text:***

The XMPL SQA organization will conduct on-going evaluations of the software development processes, work products, and software services, in accordance with the XMPL contract and SDP, to ensure:

- Adherence of the processes, work products, and services, to their applicable process descriptions, standards and procedures
- That each software product required does exist and that it has undergone software product evaluations and peer reviews, testing (where applicable), and corrective actions (for identified problems)

A detailed SQA audit schedule should be provided in the SQPP. In addition, an SQA staffing projection over the life of the program **must** be provided as illustrated in example Figure 5.16.1.



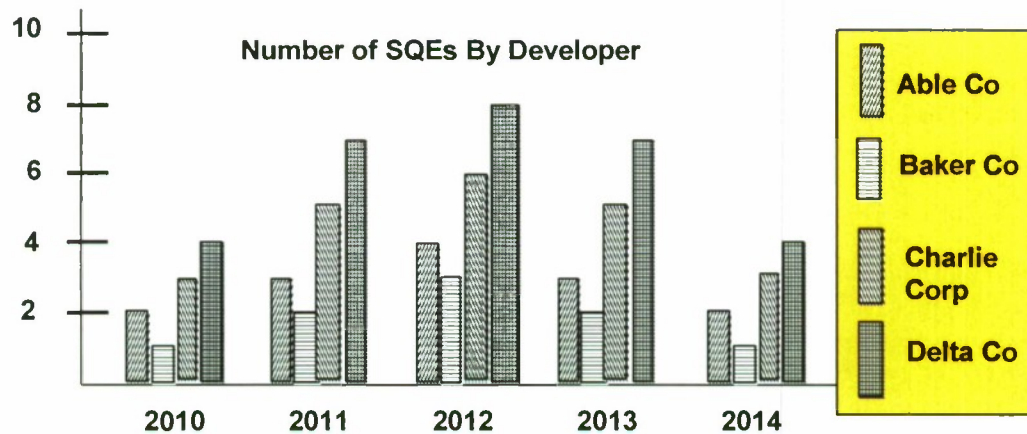


Figure 5.16.1. SQA Staffing Projection—Example

### 5.16.2 Software Quality Assurance Records, Including Items to Be Recorded

The SQA organization **must** maintain records for all evaluations performed to provide objective evidence that the evaluations were conducted. The records should consist of observations or formal findings along with their resultant corrective actions, disposition, metrics and closure. These records and reports **must** be retained in a repository for the duration of the contract and made available to the government and management as required by the contract.

### 5.16.3 Independence in Software Quality Assurance

Each SQE supports software development as an active member of their segment. However, SQEs **must** maintain a direct reporting line to their SQA organization and not be in a direct reporting line to the program they are supporting.

Independence in SQA is obtained by having a separate reporting chain to Product Assurance management. If SQA findings cannot be resolved at the lowest level possible it **must** be evaluated by the SQE to the next higher level of management. Figure 5.16.3 is an example of a program's independent reporting structure for SQA and the typical problem resolution interfaces as the resolution is elevated. The objective is always to resolve the problem at the lowest possible level.

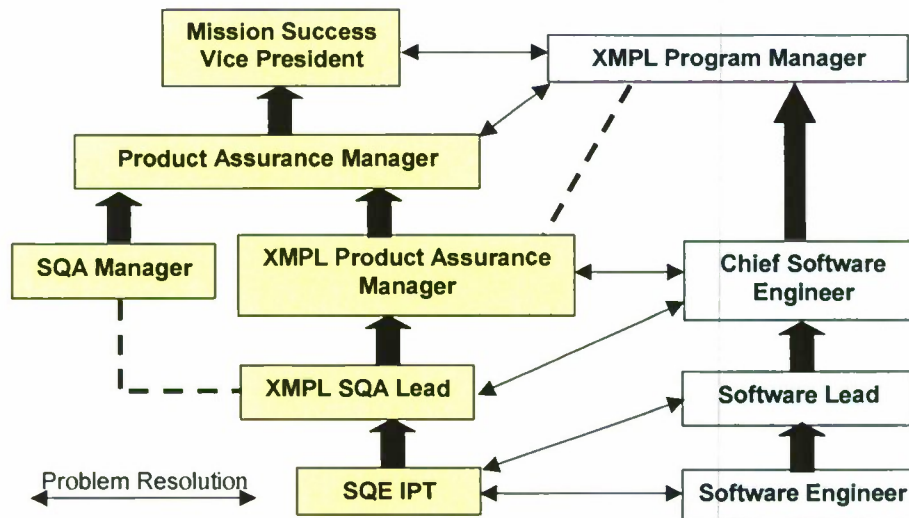


Figure 5.16.3. SQA Independent Reporting Structure—Example

In addition, this paragraph of the SDP must make it clear that the person responsible for conducting the software quality assurance evaluation must not be the person who developed, or is responsible for, the software work product. The SQE responsible for assuring compliance with the contract must have the resources, authority and organizational freedom to permit objective SQA evaluations and to initiate and verify the corrective actions.

#### 5.16.4 Software Quality Assurance Non-Compliance Issues

All noncompliance issues, identified through audits, reviews, normal SQA monitoring, ad hoc findings, etc., are candidates for corrective action or preventive action as described in subsection 5.17 of the SDP. Correction of non-compliance issues is typically handled with an automated tool, an audit database, and an established escalation mechanism to ensure that the appropriate level of management can resolve the issues. In using tools, selected by the program, non-compliance issues should be documented, tracked to resolution, and resolved within a given time frame. The Quality Assurance organization also provides metrics data to support management decision making as detailed in the Quantitative Management Plan and Software Measurement Plan.

### 5.17 Corrective Action

Corrective action is triggered when performance deviates significantly from the plan, defects are identified in the software work products, or enhancements and improvements are proposed. A definition of “significant deviation” **must** be determined by mutual agreement between the contractor and the government. The opportunity to measure progress and identify issues that need corrective action can come from the reviews and evaluations (see subsection 5.14), test results, and other quantitative management data.

In accordance with TOR 3537B, the Corrective Action activity **must** be divided into the following two paragraphs in the SDP:

- Problem/Change Reports (paragraph 5.17.1)
- Corrective Action System (paragraph 5.17.2)

#### 5.17.1 Problem/Change Reports

To report problems or changes with baselined software products, Software Discrepancy Reports (SDRs) and Software Change Requests (SCRs)—or similar names—**must** be used as part of the corrective action process. The SDR may also be called a Software Deficiency Report. The SCR/SDRs are inputs to the Corrective Action System (see paragraph 5.17.2). The difference between SCR and SDRs is:

- An SCR is typically used to enhance or improve the software product or change commitments, plans or a baseline. (It is inappropriate to classify a recommended improvement as a “problem.”)
- An SDR documents an unexpected condition or anomaly that occurs and is deemed as an incorrect action (or reaction) of the software product.

At the element/segment level, SCR/SDRs are under the control of the Software Lead but may have to be passed to a higher level board for approval. The CSWE should be responsible for SCR/SDRs at the program level. SCR/SDRs **must** be used to report a known or suspected problem or discrepancy or change with software products under any level of configuration control above the developer of the product (see paragraph 5.17.1 of TOR 3537B). The originator of the SCR/SDR should be responsible for completion of the issue description but not necessarily the person who fixes the issue.



Candidate data items for inclusion into SCRs/SDRs are: project name, originator, problem number, problem name, software element or document affected, origination date, category and severity, description, analyst assigned to the problem, date assigned, date completed, analysis time, recommended solution, impacts, problem status, approval of solution, follow-up actions, corrector, correction date, version where corrected, correction time, and description of solution implemented.

### 5.17.2 Corrective Action System

The corrective action process details can vary from location to location. These details should be documented in each developer's SDP Annex or in a Corrective Action Plan addendum to the SDP. The SCR/SDR process can also be detailed in the Software Configuration Management Plan (SCMP) and/or in lower level procedures. The division of responsibilities for corrective action tasks between the contractor team and the government's Acquisition Program Office may be confusing. A table should be included in this subsection to specifically clarify the division of responsibility. It can also be expanded on the contractor side to identify specific organizations performing each task.

Figure 5.17.2 is an example of a corrective action process overview.

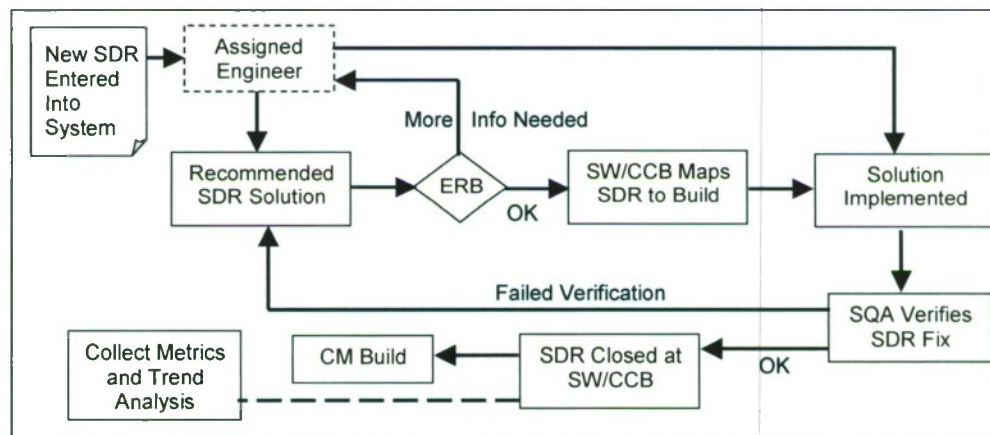


Figure 5.17.2. Corrective Action Process Overview—Example

As shown in the process example in Figure 5.17.2, once the SCR/SDR has been generated and logged at the program level, the SCR/SDR is assigned to a responsible software engineer for investigation. The investigator **must** recommend the corrective action needed and record the actions taken to either correct the problem or provide a work-around solution. When this is accomplished, the SCR/SDR is returned to the responsible Configuration Control Board (CCB) for disposition.

Once a process problem is defined, it **must** be assigned a priority and severity (see Appendix C of TOR 3537B). The status **must** be reported and tracked. The CSWE and/or SEPG should perform trend analysis on process problems and report adverse trends. Process issues are closed out when SQA verifies that the corrective action is in place and there exists objective evidence that the process is being followed. Process audit corrective action requests **must** be retained by SQA.

Corrective action measurements **must** also be collected including: SCRs/SDRs opened, closed and deferred; and aging metrics for SCRs/SDRs open for 30, 60, and 90 days, plus root cause. SCM is usually responsible for control and status updating of the SCR/SDR databases and overseeing the change management process. All SCRs/SDRs should be retained by SCM through the end of the contract.

Information concerning a non-conformity with the process **must** be forwarded to the CSWE. Trivial on-the-spot corrections do not have to be reported to the CSWE. The person reporting the process non-conformity **must** supply a problem description and a suggested corrective action. The corrective action may also be provided to the SEPG for evaluation of its overall program impact. Corrective actions **must** be evaluated to determine if: problems/issues have been resolved by someone other than the implementer; adverse trends, previously identified, have been reversed; and if changes are correctly implemented without introducing additional problems

**Configuration Control Boards:** Problems or issues for SUs with only element/segment internal interface changes, or no interface changes, should be handled with SCRs/SDRs by the segment (or element) CCB. Baselined software with external element/segment interface changes **must** be handled with SCRs/SDRs by the program level Software Configuration Control Board (SW/CCB). The program level SW/CCB has cognizance over the Product Development Baseline.

Problems or issues for SIs with no interface changes should be handled with SCRs/SDRs by the respective element's CCB. Software with external interface changes should be handled with SCRs/SDRs by the program level SW/CCB or CCB. The relationship between, and responsibilities of, these configuration control boards are discussed in subparagraph 5.14.2.2 of this Guidebook.

The following example text may be used as a partial response to this subsection:

***Example Text:***

The XMPL software Corrective Action System will use SCRs/SDRs as inputs to the system and the system will ensure that:

- All detected problems and issues are promptly reported and entered into the system
- Corrective actions will be initiated
- Status will be tracked and resolution will be achieved
- Records of the problems and issues will be maintained for the duration of the contract
- Software problems will be classified by category and severity
- Analysis will be performed to detect trends
- Corrective actions will be evaluated to determine if:
  - Problems/issues have been resolved by someone other than the implementer
  - Adverse trends, previously identified, have been reversed
  - Changes are correctly implemented without introducing additional problems

**Integration of the Corrective Action Process.** The corrective action process **must** be integrated across disciplines (software, hardware, systems engineering), IPTs, the contractor organizations (prime and suppliers), and the system development lifecycle activities (from requirements definition through system test). In addition, the corrective action process **must** be integrated with the risk management, configuration management, and the process improvement processes. For example, risks, and risk mitigation actions, can become problems that need corrective action. Process improvement actions can also create the need for corrective actions.

## **5.18 Joint Technical and Management Reviews**

Joint technical and management reviews demonstrate progress to date on project products and provide a forum for discussing programmatic issues and risks. In accordance with TOR-3537B, the Joint Technical and Management Reviews activity **must** be described by two paragraphs in the SDP:



- Joint Technical Reviews (paragraph 5.18.1)
- Joint Management Reviews (paragraph 5.18.2)

Reviews of software products and status **must** be conducted at the following levels:

- **System level:** to review system-wide project status, to identify program cost and schedule issues, and to address system technical issues (e.g., inter-segment interface problems)
- **Segment level:** to review segment-wide project status and to identify segment-specific cost, schedule, and technical issues
- **Software Item level:** to review development progress and to identify software item-specific cost, schedule, and technical issues
- **Software development level:** for feedback on in-progress technical tasks

Joint technical and management reviews **must** be conducted in concert with the Integrated Master Plan and Schedule (IMP/IMS) events and milestones. There should be a Software Review Standards addendum to the SDP defining the objectives of each type of review, the entry and exit criteria for each review, when the reviews occurs, what products are reviewed, and for what software categories these reviews **must** be conducted.

The software review process **must** be structured to support the evolution of natural products during the software development lifecycle. This should accomplished utilizing the following types of reviews:

- **Joint Technical Reviews (JTR):** conducted to review the status, correctness, and completeness of in-progress and final software products and to discuss technical issues
- **Joint Management Reviews (JMR):** conducted to demonstrate the current status of products and as a forum for discussion of status, schedules, programmatic issues, and risks
- **Technical Interchange Meetings (TIM):** similar to JTRs but are conducted in a less formal manner and may be focused on support software or specific software requirements, architecture, or design issues

#### 5.18.1 Joint Technical Reviews

Joint Technical Reviews (JTR) **must** be conducted to ensure product correctness and completeness and to elevate management and customer's visibility into the status of evolving products. JTRs focus on evaluating the adequacy and completeness of in-process or final software products. These reviews **must** be attended by persons with technical knowledge of the specific software products and have the following objectives:

- Review evolving software products using the software product evaluation criteria and guidance defined in SDP subsection 5.15.2
- Review and demonstrate proposed technical solutions; surface and resolve technical issues
- Provide insight and obtain feedback on in-progress technical tasks
- Review project status
- Surface near and long-term technical, cost, and schedule risks
- Arrive at agreed-upon mitigation strategies for identified risks, within the authority of those present, and identify risks to be raised at JMRs and to the Risk Management Board
- Ensure on-going communication between software management, developers, and the customer

The review process focuses heavily on the review and evaluation of natural products. Following the appropriate inspections and document reviews, the overall technical assessment of the product is typically presented at the JTRs, which ultimately provide software program and project status information presented at the JMRs.

Table 5.18.1 presents an example of the software product reviews by activity and the type of review to be utilized for each software category. The frequency of the reviews at each level can vary depending on the objectives of the specific reviews. When a Software Item includes a mixture of software categories, the JTR requirement **must** be at the most stringent category included in the SI. JTRs should be performed on each build so that the SSR, PDR, CDR, etc. are conducted incrementally.

Table 5.18.1. Software Product Reviews By Activity and Category—Example

Segment Reviews by Activity	MC-1	SS-1	SS-2	SS-3	C/R	SW Level	Frequency
<b>Software Requirements Definition Activity</b>							
Segment Software Specification Review (SSR)	JTR	NA	NA	NA	JTR	Seg	Once
Segment Support Software Requirements TIM	NA	TIM	NA	NA	NA	Seg	Once
<b>Software Design Activity</b>							
Software Preliminary Design Review (PDR)	JTR	NA	NA	NA	NA	SI	Once
Software Critical Design Review (CDR)	JTR	NA	NA	NA	NA	SI	SSorB
Support Software Design TIM (SSD TIM)	NA	TIM	NA	NA	NA	SI	SSorB
<b>Software Item Qualification Testing Activity</b>							
SI Qualification Test Readiness Review (SIQ TRR)	JTR	JTR	NA	NA	JTR	SI	SSorB
<b>Segment Qualification Testing Activity</b>							
Segment Qualification Test Readiness Review	JTR	JTR	NA	NA	JTR	SI	SSorB
<b>Management Reviews</b>							
Reviews of Process Compliance Audits	JMR	JMR	JMR	JMR	JMR	Sys/Seg	NA
Monthly Status Reviews	JMR	JMR	JMR	JMR	JMR	Sys/Seg	M
Program Status Reviews	JMR	JMR	JMR	JMR	JMR	Sys/Seg	Q

Type of Review: JMR = Joint Management Review  
 TIM = Technical Interchange Meeting  
 Software Level: Sys = System  
 Frequency: SSorB = Per Segment, Spiral or SI Build  
 Categories: MC = Mission Critical;

JTR = Joint Technical Review  
 NA = Not Applicable  
 SI = Software Item  
 M = Monthly Q = Quarterly  
 C/R = COTS/Reuse  
 Seg = Segment  
 SS = Software Support

## 5.18.2 Joint Management Reviews

Joint Management Reviews (JMR) **must** be periodically conducted to ensure product completeness and to elevate both management and customer visibility into the development process and evolving products. JMRs are used to review the current state of technical products, as well as project costs and schedules. Attendees **must** be persons with the authority to make cost and schedule decisions (with supporting staff) as needed.

The objectives for software JMRs include the following:

- Keep management informed about project status, directions being taken, technical agreements reached, and overall status of evolving software products
- Resolve issues that could not be resolved at JTRs
- Arrive at agreed upon mitigation strategies for near and long term risks that could not be resolved at the JTRs
- Identify and resolve management-level issues and risks not raised at JTRs



- Obtain commitments and acquirer approvals needed for timely accomplishment of the project
- Joint Management Reviews that normally apply to software are:
  - Program Quarterly Status Reviews
  - Monthly Status Reviews
  - Process Compliance Audits

The CSWE and segment IPT software personnel **must** support these reviews by providing progress-to-date and technical overviews of their software products. The SDP in this paragraph **must** define the JMRs that apply, the schedule for each, the process to be followed for each, and the personnel involved.

Specific software documentation product reviews can also be described in a table similar to Table 5.18.2 that also shows the evolution of document maturity. Table 5.18.2 is complementary to Table 3.6 in this Guidebook. Table 3.6 relates the development of software documentation to software development activities whereas Table 5.18.2 relates software documentation to formal software reviews.

Table 5.18.2. Software Documentation Maturity Mapped to Reviews—Example

Software Document	Review								
	SFR	SSR	PDR	CDR	IRR	SI TRR	PTR or BTR	EAT TRR	SQT TRR
Software Development Plan (SDP)*		B	U	U	U				
Software Metrics Report (Monthly)	P	B	U*	U	U	U	U	U	
Software Master Build Plan (SMBP)		D	P	B					
Software Requirements Specification (SRS)		P	B	U	U				
Interface Requirements Specification (IRS)		P	B	U	U				
Interface Control Document (IFCD)		B	U						
Software Design Description (SDD)			P	B					
Software Architecture Description (SAD)			P	B					
Software Test Plan (STP)			P	B	U	U	U		
Interface Design Document (IDD)			D/P	B	U				
Database Design Description (DBDD)			D/P	B	U				
Software Installation Plan (SIP)				D			P		
Software Transition Plan (STrP)				D			P		
Software User Manual (SUM)				D			P		
Firmware Support Manual (FSW)					D		P		B
Computer Programming Manual (CPM)					D		P		B
Software Test Description (STD)						D/P	B	U	
Software Test Report (STR)							B		
Software Version Description (SVD)						D	P	B	U
Software Product Specification (SPS)						D	P	B	U
SDP Subsection:	5.4	5.5	5.6.2	5.6.3	5.7	5.8	5.9	5.10	5.11

D = Draft    P = Preliminary    **B** = Baseline    U = Updated (As Required)

SFR = System Functional Review

SSR = Software Specification Review

PDR = Preliminary Design Review

CDR = Critical Design Review

EAT TRR = Element (or Factory) Acceptance Test – Test Readiness Review

SQT TRR = System Qualification Test – Test Readiness Review

IRR = Integration Readiness Review

SI TRR = Software Item – Test Readiness Review

PTR or BTR = Post Test or Build Turnover Review

\*Draft with Proposal; Preliminary at ATP+ 60-90 days

## 5.19 Software Risk Management

Software risk management **must** be a continual process employed throughout the software development lifecycle. Risks are defined items that may cause a significant deviation from accepted performance criteria. Software risk management addresses the management process for identification, mitigation, and tracking of software development risks that involve potential adverse technical, programmatic, schedule, cost, or supportability impact throughout the software development process.

There is no specific organization required for this subsection by J-16 or TOR-3537B. Figure 5.19 is an example of an overview of the risk management process.

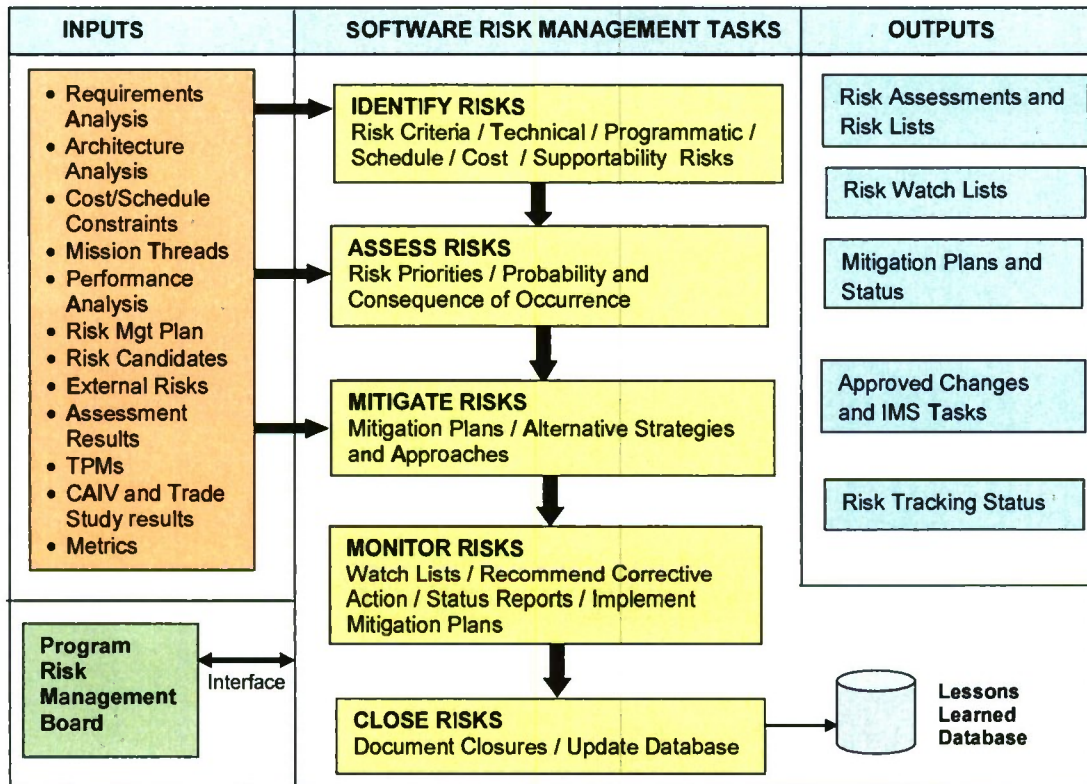


Figure 5.19. Risk Management Process Overview—Example

Software risk management provides direction to ensure that the project makes an early and continuing identification of its top software risk items, develops a strategy for handling the risk items, identifies, and sets down an agenda to handle new risk items as they surface, and highlights progress versus plans for risk items.

All risks **must** be handled in some manner. Handling risks includes mitigation, where additional resources are spent to reduce the likelihood of the risk happening, or mitigation to reduce the severity of the risk if it does happen. Some risks are never resolved. The risk might “go away” over time—an example is the risk a COTS vendor will go out of business. Risks that are not candidates for mitigation **must** still be watched via some tracking metrics so that changes in that risk are known by management in a timely manner.

**Objective.** The objective of the software risk management process is the development of a mechanism for regular monitoring and management of software development risks through the effective utilization of a Risk Management (or Mitigation) Plan (RMP). Details of the software risk



management approach and process **must** be contained in the RMP. The Risk Management Board (RMB) is the primary entity for evaluating risks, unfavorable event indications, watch list items, and concerns. The CSWE should be a member of the Risk Management Board.

For most programs, the RMP is a program-level plan—not a software plan. Similarly, the RMB is at the program level. The software risk management process described in this subsection of the SDP **must** be consistent with the program risk management process (described in the RMP) and **must** interface with the process for elevation of software risks of sufficient concern.

However, there will likely still be software risks that are not “big” enough to make it to the program risk list. These risks **must** be managed using the software risk management process described in this subsection. Typically, there is no RMB at the software level, but there is nothing to preclude it.

During software development, software risk management involves identifying, assessing, documenting, and mitigating risks. Individual risk plans define mitigation tracking measures and corrective action when thresholds exceed the limits defined in these plans.

**Goal.** The goal of risk management is to reduce or eliminate, early in the program and throughout the program, potential problems that could adversely affect technical, programmatic, schedule, cost, or supportability performance. The RMP identifies the process for risk planning, identification, assessment, prioritization, handling, and monitoring. A Risk Handling Plan (RHP) may be generated for any identified risk and is the responsibility of the affected IPT. RHPs **must** be approved by the RMB. IPT leads are accountable for implementing the RHP and reporting risk status within scope of their CWBS elements.

**Approach.** The software RMP, typically an Addendum to the SDP, is the program’s plan for identifying software risks and mitigating the risks as necessary. The software RMP should assign a risk severity levels, define risk handling plans where appropriate, and describe the process for ensuring implementation of the risk handling plans. It also provides the program team’s plans for maintaining and improving the software process capability maturity of the software team members throughout the life of the program.

The program Software Metrics Plan (SMP) describes software metrics used to control and track risks. The SDP needs to be clear as to what risks are handled at the program level and what is handled at the software level. Risks are integral to software development, managed, and coordinated across segment and development tasks. The overall software risk mitigation approach can be summarized as follows:

- Software build planning **must** be consistent with the Software Risk Management Plan
- The risk handling approach encourages cross-support of software engineering process groups particularly for process improvement and risk reduction (see subsection 5.25)
- Risk assessment is integral to each review (the CSWE’s oversight of software tasks across the program helps reduce software risk by resolving issues early)
- Incremental software development mitigates risk. For each increment, technical, programmatic, schedule, cost, or supportability issues and requirement changes are assessed, baselined, prioritized, tracked, and resolved early in development
- Prototypes can evaluate hardware and software integration, and multi-contractor development and integration. These tools are used as an integration and demonstration facility to evaluate risks early in system design and development
- Metrics and critical path analysis are integral to risk management and provide guidance for reducing, preventing, or eliminating adverse impacts

## 5.20 Software Management Indicators

There is no specific outline required by J-16 or TOR-3537B for software management indicators (also referred to as 'software metrics'). The recommended organization for this subsection is based on the Air Force Space Command, Space and Missile System Center (SMC) Instruction 63-104, dated 21 November 2005. Section 2.10 of that Instruction "Metrics, Assessment, and Improvement" requires that each SMC program office shall:

"...describe how they set and use metrics objectives and thresholds. Include objectives, thresholds, plans, actuals, and historical data in managing the acquisition, development, and sustainment (if applicable). The description shall delineate how the metrics are used to influence program decisions."

That is what the program office needs, so the plans to manage software development using software measurements **must** be included in either the SDP, a Software Measurement Guidebook addendum to the SDP, the Quantitative Management Plan (QMP) that may be an IMP appendix—or in all three at appropriate levels of detail.

It is permissible to include a few short introductory paragraphs in subsection 5.20 and refer to the program's Software Measurement Guidebook or QMP for the details. However, since the use of software management indicators is so important, it is recommended to include an overview of the measurement approach in the SDP similar to the example paragraphs below.

Software measurement initiatives **must** be in accordance with the software measurement standard imposed on the program as well as:

- Existing contractor organizational and contractually imposed software measurement policies, standards, and procedures
- The international standard: ISO/IEC 15939-2002, *Software Engineering—Software Measurement Standard*
- *Practical Software Measurement: Objective Information for Decision Makers*, McGary, et al, Addison Wesley, October 2001

Each software measurement initiative **must** be tailored to each individual program. The software development team **must** use software management indicators to aid in managing the software development process and communicating the status of the software development effort to the customer, program management, software management, and software personnel. Management indicators are critical to the software management process because effective management controls are dependent on timely and accurate measurements.

### 5.20.1 Principal Objectives of Measurement

Typical objectives of a software measurement initiative are to provide:

- Relevant and timely information to help software leads and software engineers perform their responsibilities correctly, on-time, within budgets, and to produce a higher quality product
- Tracking information for project management to facilitate the reduction of the project's software cost, schedule, and technical risks



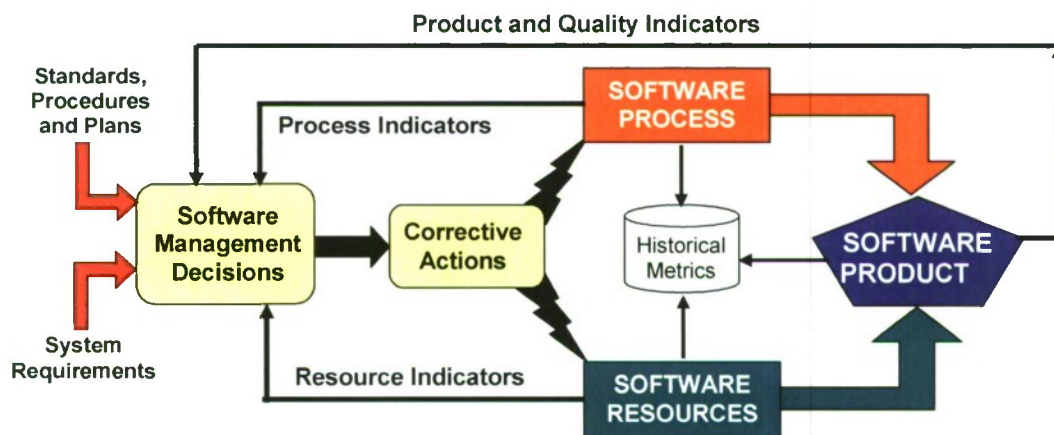
- A practical, efficient and up-to-date management methodology and basis for quantitative software development control, status determination, timely corrective action and activity replanning.
- Historical records of performance for trend analysis and other value-added information, to support continuous software process improvement

### 5.20.2 Continuous Improvement

A fundamental aspect of the software measurement initiative should be a continuous improvement approach through a closed loop feedback control system as depicted by the example in Figure 5.20.2. Feedback information **must** be provided so that corrective actions can be applied to improve the development process, maximize resource utilization and predict and adjust the quality of the products.

Timely corrective action **must** be taken to realize the benefits of a software metrics program. It should not become merely a historical archive. However, historical information should be used for trend analysis, productivity calculations, and continuous process improvement.

Figure 5.20.2 depicts the essence of the SEI Software Process Maturity Level-4 Key Process Areas. It is an important model to use as a goal. However, the simplicity of this diagram belies the inherent complexity of achieving a smoothly running, unencumbered, skillfully managed closed loop software development control system.



Software management indicators are measurable attributes of the software development process, the resources applied and the products produced—the process and the resources produce the product.

Figure 5.20.2. Closed Loop Software Control Process—Example

### 5.20.3 Approach to Management Measurements

An example of a top-down management measurement approach is depicted in the framework shown in Figure 5.20.3. It shows a tailorable hierarchy of four groups. Multiple users may have similar objectives but each has a different perspective and information needs. The effort to collect, analyze, and document metrics **must** be consistent with their value to the program.

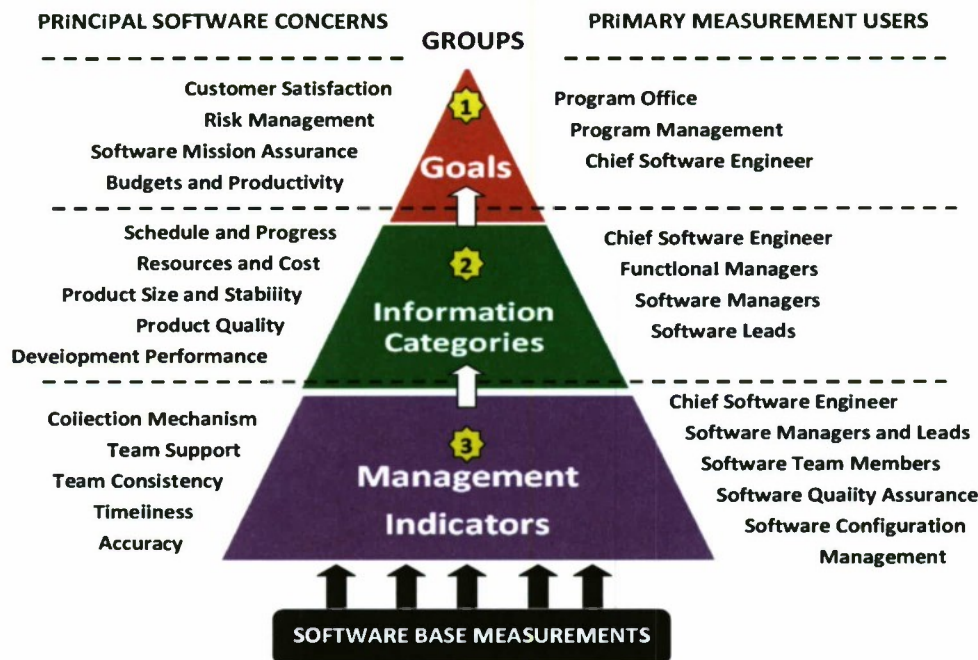


Figure 5.20.3. Software Measurement Framework—Example

**GQM Paradigm.** This example of a top-down approach is based on the Goal-Question-Metric (GQM) paradigm. At the top of Figure 5.20.3, software management establishes program/project goals (Group 1) resulting in a set of Measurement Categories (Group 2) to determine progress in meeting the goals. The categories identify a set of Measurement Indicators (Group 3) that provides support to the Group 2 categories. The indicators are composed of detailed software base measurements (Group 4) that **must** be collected to provide the data needed by the indicators. Following is a brief description of the four levels.

- **Goals (Group-1):** Top-level Group-1 program/project goals are of primary interest to senior managers, the customer, and the CSWE. They provide an effective means to appraise and track software milestones and overall project trends. Data to support the goals can be derived from a combination of metrics collected and calculated from lower groups.
- **Software Information Categories (Group-2):** The Group-2 software categories address the question: What information do software managers, leads, and developers need to manage their task in a timely and effective manner and be responsive to program/project goals?
- **Software Management Indicators (Group-3):** Group-3 focuses on specific software management indicators that **must** be collected to support the Information Categories in Group 2.
- **Software Base Measurements (Group-4):** Specific measurements (raw data) **must** be collected to collect the data needed for the Management Indicators in Group-3. (For example, to collect the monthly “requirements volatility” management indicator, it is necessary to collect measurements of the number of requirements added, deleted, and modified this month plus the total number of active requirements last month). Significant changes should be reported as they occur, and formally reported at the next reporting period.



### 5.20.4 Key Software Management Questions

Figure 5.20.4 lists five key questions that Software Leads and Managers **must** have periodic responses to effectively manage the software development effort. There is no restriction to adding more key issues if needed but the five questions listed should constitute a minimum set. Figure 5.20.4 also lists the five Information Categories that directly support answers to the five key questions. The five categories are supported by specific Measurement Indicators as shown in Figure 5.20.4.

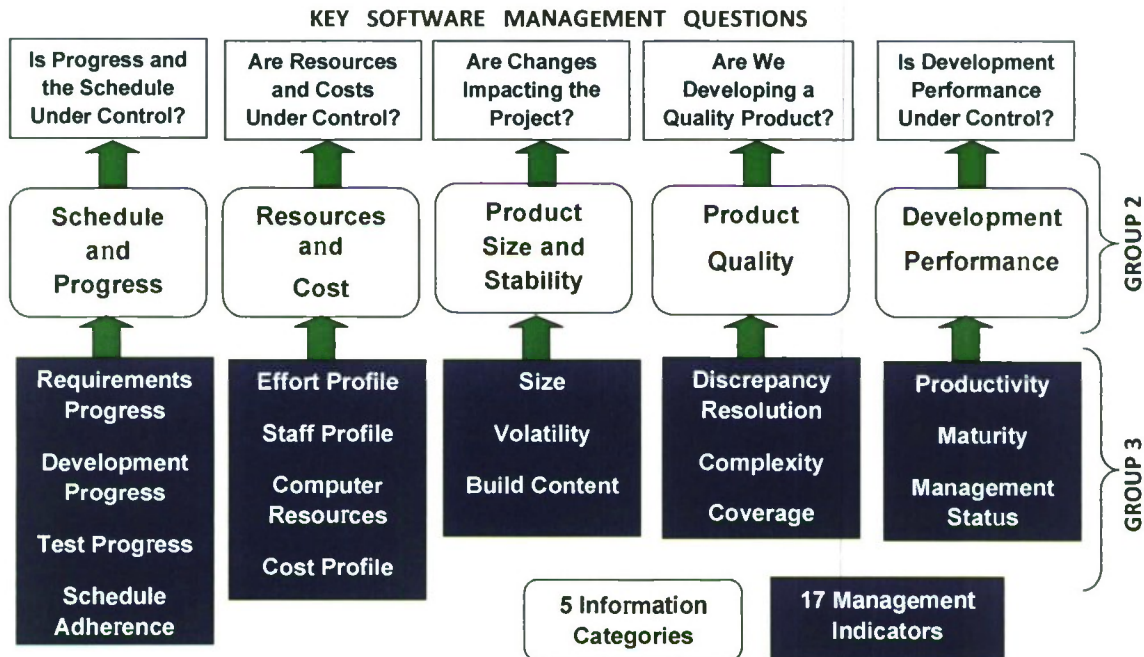


Figure 5.20.4. Categories and Indicators Support the Key Management Questions—Example

### 5.20.5 Software Measurement Set

Table 5.20.5 contains an example software management set. It includes the three levels (Information Categories, Management Indicators, and Base Measurements) consistent with Groups 2, 3, and 4 as shown in Figure 5.20.3. It is highly recommended that a table of this nature be included in the SDP or in a Software Measurement Guidebook that should be an addendum to the SDP.

An optional candidate list of software management indicators is provided in Appendix F of TOR 3537B. In addition, Chapter 14 of the SMC Software Acquisition Handbook, dated 9 Feb 2004, contains a description of recommended software indicators with a clear understanding that they should be tailored to the system being developed and that additional indicators should be added to address critical or unique needs of each program.

### 5.20.6 Software Measurement Construct

A software measurement construct defines the data that will be collected, the computations that will be performed on that data, and how the resulting data will be reported and analyzed. The recommended construct for software measurement is described in detail in the *Software Measurement Standard for Space Systems* (SMSSS), The Aerospace Corporation report number TOR-2009(8506)-6, dated May 5, 2011. As shown in Figure 5.20.6, the software measurement construct is composed of four specifications:

- Measurement Information Specification
- Base Measurement Specification
- Derived Measurement Specification
- Measurement Indicator Specification

Table 5.20.5. Software Measurement Set—Example

Software Information Categories	Management Indicators	Base Measurements
Schedule and Progress	Requirements Progress	Requirements Defined
		Requirements TBX Closure
		Requirements Verified
		Qualification Method
	Development Progress	Components Defined
		Units Defined
		Unit Coded and Unit Tested
		Unit Integrated and Tested
	Test Progress	Test Cases Developed
		Test Cases Dry Run
		Test Cases Performed
		Test Cases Passed
	Schedule Adherence	Project Milestones
		Scheduled Activities
Resource and Cost	Effort Profile	Labor Hours by Activity
		Rework Hours by Activity
	Staff Profile	Staffing Level
		Staff Experience
		Staff Turnover
	Computer Resources	CPU Utilization
		Input/Output Utilization
		Memory Utilization
	Cost Profile	Response Time
		Earned Value Performance
Product Size and Stability	Size	Schedule and Cost Performance Index
		Schedule and Cost Variance
		Requirements Size
		Requirements by Type
	Volatility	Line of Code Size
		Line of Code by Origin
	Build Content	Line of Code by Type
Product Quality	Discrepancy Resolution	Requirements Volatility
		Line of Code Volatility
		Requirements per Build
		Discrepancy Report Status
	Complexity	Discrepancy Report Aging
		Discrepancy Report by Type
Development Performance	Coverage	Discrepancy Report by Source
		Cyclomatic Complexity
	Productivity	Requirements to Design Traceability
	Maturity	Requirements to Test Case Traceability
	Management Status	Development Productivity
		Development Defect Density
		Action Items Closure
		Risk Mitigation Task Status
		Schedule Compression



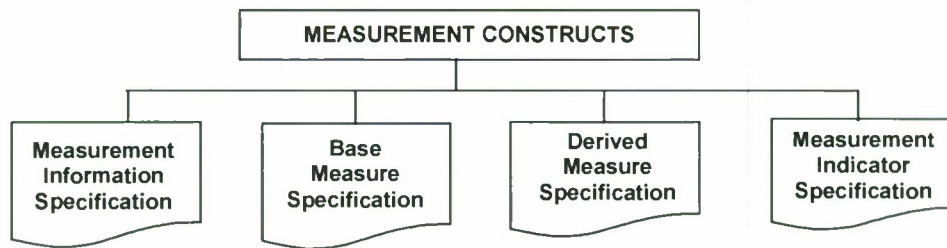


Figure 5.20.6. Elements of the Software Measurement Construct—Example

The data required by these specifications should be included in the Software Measurement Guidebook that is normally an addendum to the SDP. The four specifications are briefly described below.

**Measurement Information Specification.** Each Management Indicator shown in Table 5.20.5 is specified using a Measurement Information Specification. Table 5.20.6-1 is the format for this specification used to describe the information need and the measurable concept to address that need. The template also contains fields for indentifying relevant entities as well as the base and derived measures that implement the information need.

Table 5.20.6-1. Format of the Measurement Information Specification—Example

Name	Measurement Description
<b>Information Need</b>	What the measurement user (e.g., manager or project team member) needs to know to make informed decisions.
<b>Information Category</b>	A logical grouping of information needs provided to give structure to the measurements. The five recommended information categories are: schedule and progress, resources and costs, product size and stability, product quality, and development performance.
<b>Measurable Concept</b>	Satisfying the information need by defining the data to be measured.
<b>Relevant Entities</b>	The objects to be measured. Entities include process or product elements of a project such as project tasks, plans/estimates, resources, and deliverables.
<b>Base Measure</b>	The property or characteristic of the data that is quantified.
<b>Derived Measure</b>	A measure that is calculated as a function of two or more base measures.

To help clarify how the Measurement Information Specification is used, Table 5.20.6-2 is an example taken from the SMSSS that shows how it can be applied to the 'Staff Profile' Management Indicator.

**Base and Derived Measure Specifications.** Table 5.20.6-3 is an example format for the Base and Derived Measurement Specifications. Some base and derived measures are used to define multiple measurement constructs. Not all measures require the specification of a derived measure. The appendixes in the SMSSS contains detailed descriptions of the base and derived measurements including where and how they are obtained, how often reported, the scale, and unit of measurement.

Table 5.20.6-2. Example of a Measurement Information Specification for Staff Profile

Measurement	Staff Profile
Information Need	Evaluate staffing requirements to see if staffing assumptions are being realized.
Information Category	Resources and Cost
Measurable Concept	<ul style="list-style-type: none"> <li>Compare planned staffing requirements to actual staffing provided to determine staffing status.</li> <li>Compare planned experience to actual experience to identify staffing competency shortfalls.</li> <li>Compare staffing gained and lost to plan to identify staffing trends.</li> </ul>
Relevant Entities	Planned Headcount Actual Headcount
Base Measures	<ul style="list-style-type: none"> <li>Planned Head Count—Total</li> <li>Planned Head Count—Experience Level Category 1</li> <li>Planned Head Count—Experience Level Category 2</li> <li>Planned Head Count—Experience Level Category 3</li> <li>Actual Head Count—Total</li> <li>Actual Head Count—Lost</li> <li>Actual Head Count—Gained</li> <li>Actual Head Count—Experience Level Category 1</li> <li>Actual Head Count—Experience Level Category 2</li> <li>Actual Head Count—Experience Level Category 3</li> </ul>
Derived Measure	The following derived measure is used to graph these indicators: <ul style="list-style-type: none"> <li>Staffing Volatility Index</li> </ul>

Table 5.20.6-3. Base and Derived Measure Specifications—Example

Measurement	Base Measure Specification
Base Measures	A measure of a single attribute defined by a specified measurement method (e.g., planned number of lines of code).
Measurement Methods	The logical sequence of operations defining the counting rules for each base measure.
Type of Method	Method used to quantify an attribute as either (a) subjective, involving human judgment or (b) objective, using established rules to determine numerical values.
Scale	The ordered set of values or categories that are used in the base measure.
Type of Scale	The type of the relationship between values on the scale, either: <ul style="list-style-type: none"> <li>Nominal—the measurement values are categorical, as in defects by their type</li> <li>Ordinal—measurement rankings, as in assignment of defects for severity levels</li> <li>Interval—measurement values having equal increments</li> <li>Range—a range of real numbers for equal quantities of the attribute</li> </ul>
Unit of Measurement	The standardized quantitative amount that will be counted to assign value to the base measure, such as an hour or a line of code.
Measurement	Derived Measure Specification
Derived Measures	A measure that is calculated as a function of two or more base measures.
Measurement Function	The formula that is used to calculate the derived measure.

**Measurement Indicator Specification.** Table 5.20.6-4 is an example format for the Measurement Indicator Specification. The first part of this specification includes a description and a sample display diagram. The purpose of the sample display diagram is to show the preferred format for the information depiction. The specification also includes fields for analysis guidance and the decision criteria that triggers a set of actions in response to specific threshold values.



Table 5.20.6-4. Format for the Measurement Indicator Specification—Example

Measurement	Measurement Indicator Specification
Indicator Description and Sample Display Diagram	A description and display of one or more measures (base and derived) to support the user in analysis and decision making. An example diagram of the indicator is included.
Analysis Model	A process that defines the responses of the measurement user to the indicators. If decision criteria are specified this field describes their use.
Decision Criteria	A defined set of actions that will be taken in response to specific values of the indicator.
Additional Analysis Guidance	Any additional guidance on variations of this measure.
Implementation Considerations	Any process or implementation requirements necessary for a successful implementation.

### 5.20.7 Analysis and Reporting of Software Management Indicators

Software metrics data **must** be reported to Program Management and the customer at least monthly to provide frequent status checks of the development effort. When potential problems are identified, the affected Software Lead **must** analyze the problem indicators to determine if the data accurately reflects a real or developing problem and, where necessary, if timely management action is needed to correct the problem. Coordination with the Integration and Test team may be necessary for support in analyzing the potential problem or developing and implementing a correction.

The SEPG should review the software indicators to determine their effectiveness. Software indicators should be added and deleted as their utility and cost/benefit is determined. The following questions should be considered:

- Is the metric providing needed information to software IPTs in sufficient time to implement management actions to minimize cost or schedule impacts?
- Does the metric accurately measure the software development process activity it is intended to measure and provide a meaningful status?

A recommended best practice is to establish and conduct a joint customer—contractor Software Measurement Working Group to ensure the information needs of all participants are met as the development effort proceeds through the lifecycle.

The SEPG at the program or segment level **must** perform analysis of the software management indicators in accordance with their scope of control. The SEPG, or the responsible working group, should meet at least monthly to review the results of the metrics analysis and report their findings to the segment IPT and the CSWE. Findings should also be distributed to the Software Leads for their use in managing their daily tasks. The CSWE should report results to program management at regular intervals.

**Quantitative Management Plan (QMP).** The QMP, if one is prepared, should define the establishment of program goals as well as the methods used for collection, analyzing, quantitatively controlling, and reporting performance data in terms of the goals. In addition, it should present strategies for achieving the goals, performing causal analysis and determining potential corrective actions. Understanding the root cause of a problem is an effective path to preventing it.

**Reporting Durations.** The SDP should also include a table indicating when in the lifecycle the collection of each software management indicator is started and when it ends as well as the reporting intervals. This data collection duration table is important since all of the measurements are not collected all the time (e.g., testing measurements are not collected during requirements analysis). The SMSSS (referred to in paragraph 5.20.6 above) contains a detailed table with time phasing for all base measurements.

### 5.20.8 Software Indicator Thresholds and Red Flags

**Software Thresholds.** Management decisions based upon the analysis of software management indicators **must** use thresholds to flag non-nominal conditions. Thresholds **must** be established by the responsible working group for the segment IPT. When a value is outside the nominal conditions, the segment IPT should determine:

- How the problem can be fixed to bring the values within the stated limits
- If the out-of-limit condition is an acceptable design-related decision

Table 5.20.8-1 is an example format of allowable thresholds for some of the software indicators. A column containing thresholds could be added to the measurement set in Table 5.20.5.

Table 5.20.8-1. Software Indicator Thresholds—Example

Software Information Categories	Software Management Indicators	Threshold
Schedule and Progress	Requirements Progress	±10% From Plan
	Development Progress	±10% From Plan
	Test Progress	±10% From Plan
	Schedule Adherence	±10% From Plan
Resources and Cost	Effort Profile	±10% From Plan
	Staff Profile	±10% From Plan
	Computer Resources	±10% From Plan
	Cost Profile	±10% From Plan
Product Size and Stability	Size	±10% From Plan

**Software Indicator Red Flags.** Potential problems with software management effectiveness can be anticipated at the time the RFP, or contract, is issued if specific requirements for the measurement program are not identified. For example, the contractor and the Program Office should be aware of red flags such as the examples listed in Table 5.20.8-2.

Table 5.20.8-2. Software Indicator Program Red Flags—Example

RED FLAG if the RFP, or contract, does not require:
Formal delivery of periodic measurement reports and analysis of software management indicator data
A Software Measurement Guidebook, or Quantitative Measurement Plan, delivered with the proposal
Measurement data to be presented in a useful, unambiguous, and easy to read graphical form
Definition of measurement collection durations
A restriction on the frequency of report format changes
Measurement commitments to be flowed down to subcontractors
Documented allowable indicator threshold deviations
A documented plan to determine root cause for deviations
A clear plan for corrective action when deviations are identified
Sufficient measurements to effectively monitor program status (Note: an excessive number of indicators may also be a red flag)

### 5.21 Security and Privacy Protection

There is no required format for this SDP subsection. Each facility in the program **must** generate its own security standard practices and procedures document to cover security-related specifics for that facility. These security practices and procedures documents should include at least the following items:



- Physical safeguards employed to control access to the development and test environments during and after classified processing
- An outline of the startup or upgrade procedures for classified processing; the safeguards employed during classified processing, including the security features and assurances of the software and the trusted operating system, or equivalent; and the procedures for shutdown or downgrading the system
- Accountability procedures for the control and dissemination of classified materials and a description of the procedures for declassification and destruction of storage media
- A description or exhibits of pertinent automated or manual audit trail records and logs
- A contingency plan to be employed in the case of security violations, system crashes, or other emergencies during classified processing, including recovery procedures. Clearances and declassification procedures should be emphasized, as appropriate

The IPT software personnel **must** periodically review requirements for security and privacy contained in the program's Security Implementation Plan. Also see SDP subparagraph 4.2.5.2 that addresses the strategy for handling critical security requirements.

## 5.22 Subcontractor Management

There is no required SDP format for the subcontract management subsection. From a software development perspective, the management of software subcontractor teams is almost always a significant challenge. There are usually numerous subcontractors contributing software products, and they are typically geographically dispersed. Lessons Learned should influence the prime contractor's development of the Statement of Work (SOW) for each teammate so as to avoid problems encountered in previous software subcontract management efforts.

**Subcontractor Management Team.** A Subcontractor Management Team (SCMT) **must be established**. It should be led by the Subcontract Program Manager who has overall responsibility for monitoring technical, schedule, and cost performance of the software subcontractors. A Software Subcontract Management Guidebook should be prepared. The Software Quality Engineer (SQE) and the Chief Software Engineer (CSWE) **must** support subcontract management by technically monitoring the software portion of the subcontract. Table 5.22 is an example of typical SCMT membership and their responsibilities.

**SDP Compliance.** The CSWE should be responsible for monitoring software subcontractors through attendance at the software subcontractor's formal reviews and status reviews, as applicable, and by regular metrics reviews. The CSWE **must** evaluate the performance of the software subcontractors and prepares subcontract evaluation reports as required. The CSWE can delegate these responsibilities.

The program-level SDP **must** apply to all software subcontractors. The CSWE has the authority to enforce the processes described in the program-level SDP. Software subcontractors **must always** follow the processes, procedures, and documentation defined in program-level SDP. As described in Part 1: Introduction Section 9, site-specific SDPs are written and maintained by the development sites and provide additional standards and procedures specific to each site. Site-specific SDPs expand upon, but **must not** conflict with, the processes and procedures defined in the program-level SDP unless a waiver has been approved. The CSWE and SQE **must** perform software subcontractor product and process audits to determine compliance with program-level SDP and with the contract.

Table 5.22. Subcontractor Management Team Members and Responsibilities—Example

SCMT Member	Responsibilities
<b>Subcontract Program Manager</b>	Overall management of software subcontract technical, cost, and schedule performance Ensures all software technical, cost, and schedule requirements are satisfied Facilitates subcontractor's ability to plan and perform software more efficiently Works program-level issues and manages software award fee program Ensures subcontractor compliance with program plans and procedures
<b>Subcontract Administrator</b>	Single point of contact for contractual matters and administers the software subcontract Negotiates and awards software subcontracts and approves vouchers/invoices Ensures proper flow down of software technical requirements and software subcontract terms and conditions Maintains configuration control of contractual documentation sent to the subcontractor Receives, logs, and distributes incoming correspondence from subcontractors
<b>Responsible Engineer</b>	Develops software specifications and associated technical documentation Ensures subcontractor understanding of the software technical requirements Coordinates approval of subcontract software deliverables and documentation Conducts a technical evaluation of subcontractor's software proposals Develops and/or approves software test plans and procedures and acceptance plans Participates in or witnesses subcontractor software acceptance testing, as required Provides independent evaluation of subcontractor's technical progress and performance
<b>Business/Financial Operations</b>	Analyzes cost/schedule performance data including key indices and variance analyses Helps subcontractor develop cost account plans and incremental planning packages Integrates subcontractor budgets, costs, IMP, and IMS into Prime's database system
<b>Mission Assurance</b>	Monitors data, configuration, and quality processes used by the subcontractor Ensures the software subcontract quality implementation program is consistent with the program's SQPP Ensures that the configurations of all deliverable software items are identified with a clear audit trail Chairs the Subcontractor Functional and Physical Configuration Audits (FCA/PCA)

**Performance Review.** The progress and performance of the subcontractors should be regularly reviewed and assessed by the SCMT. These reviews address the total performance of each team toward meeting its objectives. The SEPG oversees the software development process and provides approval of all subcontractor specific appendices to the program-level SDP to ensure that software development methods, standards, practices, and procedures are consistent with the contract.

**Statement of Work.** The subcontractor SOWs should be focused on a clear definition of responsibilities, software metrics reporting, interfaces with the SCMT for oversight, and the CSWE as the single point of contact for software within the organization. Each team member performing software development, algorithm analysis, simulation development, or data set development should have its own SOW. The SOW should delineate development software products, scope, required reviews, schedule milestones, status reporting, performance evaluation criteria, and acceptance criteria.

The SCMT should be responsible for ensuring SOW compliance, flow-down of requirements changes, updates to the SOW, and for assuring that other applicable documents (specifications, interface, or performance documents) are current and available to the subcontractor.

### 5.23 Interfacing with Software IV&V Agents

There is no required format for this SDP subsection. If required on the contract, software Independent Verification and Validation (IV&V) agents should interface with segment software development as a member of their respective Integrated Product Teams (IPTs). In addition to supporting software-related IPT tasks, they may also perform audits of software development files and software processes.

If audits are performed, the IV&V agent **must** coordinate in advance with the IPT lead and identify required developer and test support. Additional details for interfacing with software IV&V agents may be defined in the SDP Annexes.



IPT software personnel, including developers and test engineers, should interface with the IV&V representatives to allow identification and resolution of software issues and problems at the lowest practical level and as early in the development process as possible. The IV&V representative **must** interface with the software IPTs and developers, both formally and informally, through the following mechanisms:

- At scheduled software development status meetings, reviews, Technical Interchange Meetings (TIMs), and IPT meetings
- Through the test process, either when the IV&V representative is a test witness or when the IV&V representative conducts independent testing
- Through periodic inspections and audits of Software Development Files (SDF), other software products, and the software development process

Table 5.23 is an example of the software products that **must** be provided to the IV&V agents and how problems are reported. Problems identified by the IV&V agents **must** be handled through the corrective action process (see SDP subsection 5.17).

Table 5.23. Software IV&V Evaluations—Example

Tasks	Software Products Reviewed	Problem Reporting Mechanism
TIMs	Development phase documents	Action items
Formal software reviews	Development phase documents	Action items
Audits	SDFs, SDP	Audit report
Independent testing	Software test documents	SCR/SDRs

IV&V representatives should be members of appropriate software IPTs. They should offer advice, assistance, and subject matter expertise in the development of program documentation, but must not co-author such documentation. This allows the IV&V representatives to preserve the degree of objectivity required to effectively discharge the IV&V role of verifying and validating both adherence to the software development process, and the adequacy, sufficiency, and performance of software products.

## 5.24 Coordination With Associate Developers

The lead prime contractor should enter into what might be called “Associate Contractor Agreements” with other prime contractors whose performance will impact the program. Such arrangements facilitate joint participation and collaboration in meeting program requirements. The objective of these agreements is to create ground rules and an environment for freely sharing and safeguarding each other’s technical and/or proprietary information and resolving issues, to the maximum extent possible, without government intervention. **As they become known, the lead prime contractor must advise the other prime contractors, in addition to the organizations that may be defined in the Model Contract, where such agreements are necessary and then proceed with establishing those relationships.** From a software perspective, Associate Developers are one type of software stakeholders.

## 5.25 Improvement of Project Processes

There is no required format for this SDP subsection, however, **subsection 5.25 should cover a general description of the tasks planned for identifying software process improvement areas and developing new process policies and procedures to implement process improvements.** It should also contain an overview of the software process improvement process. Figure 5.25 is an example of a graphically depicted process improvement process.

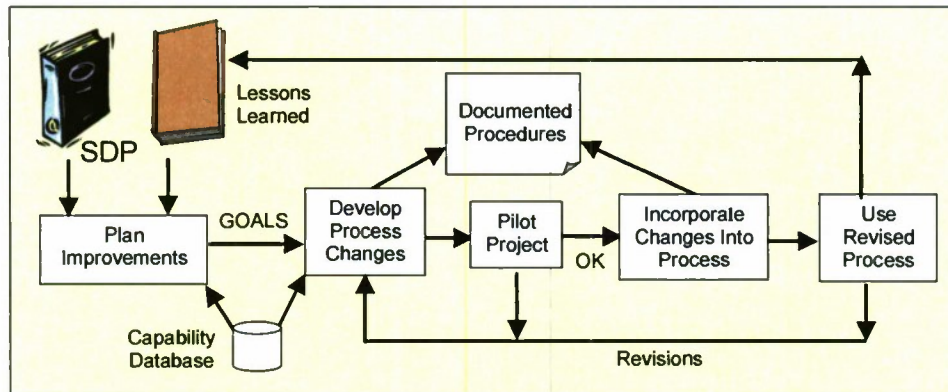


Figure 5.25. Software Process Improvement Process Overview—Example

### 5.25.1 Software Engineering Process Group

A Software Engineering Process Group (SEPG), or an equivalent organization, **must** be established to focus on:

- Evaluating the implementation and progress of the defined software development process
- Identifying areas for potential process improvement
- Performing evaluations to determine if the deficiencies are the result of non-compliance with, or inadequacy of, current policies and procedures

If the deficiency is non-compliance, the SEPG **must** determine the nature of the non-compliance and either identify a process improvement or recommend to the CSWE how to correct the compliance deficiency as described by the corrective action process. If an inadequacy, the SEPG **must** analyze the affected process area and develop proposed process improvements with corresponding changes to the SDP. Table 5.25.1 contains the typical membership of the SEPG and their responsibilities.

Table 5.25.1. SEPG Membership and Responsibilities—Example

SEPG Membership	Responsibilities
Chief Software Engineer	The CSWE is the SEPG Chair and: (1) Ensures SEPG tasks are assigned and performed in a timely manner; (2) Allocates resources for the SEPG, including people, time and equipment; and (3) Encourages and actively supports SEPG tasks.
Software Process Engineer	The SPE is typically the SEPG Administrative Chair and: (1) Ensures the SEPG meetings are organized (agenda, facilities, etc) and meeting results are recorded (meeting minutes) and tracked (Action Items, Working Group status); (2) Participates as the expert on software processes and software process improvement methods; and (3) Is the point of contact for software process improvement issues and SEPG tasks.
Software Configuration Management	The SCM Lead: (1) Participates as the expert on SCM processes and procedures; and (2) Is the point of contact for SCM process improvement issues.
Software Quality Assurance	The SQA Lead: (1) Is the point of contact for SQA issues; and (2) Audits SEPG tasks to assure conformance with contractual requirements, plans, standards and procedures.
SEPG Members	Each organization responsible for a software product assigns member(s) to represent the organization as their SEPG Representative and who are the organization's point of contact for software process improvement and SEPG tasks.

**SEPG Focus.** The focus of the SEPG is to: assess the current process status; define, document, maintain, monitor, and improve the program software process; establish software training requirements; establish a program-specific archive (often called the SPAR—Software Process Assets



Repository); and work to improve the program's software CMMI rating. An example of how to focus the software process improvement initiative, in terms of goals, approach and the measure of success, is shown in Table 5.25.2.

Table 5.25.2. Focus of the Process Improvement Initiative—Example

Area	Segment IPT	Program SEPG
Goals	Produce and maintain quality software and satisfy system requirements.	Perform continual process improvement through quantitative feedback from the process and from innovative process improvement tools.
Approach	Use the most effective software engineering tools and techniques including metrics collection.	Perform measurement analysis, assess processes, maintain a library of experiences, and create and update software standards.
Measure of Success	Delivery of quality software products on time and within budget.	Improve processes to result in improved products, reuse growth, and efficient collection, storage, and retrieval of experiences.

**SEPG Mechanisms.** The SDP should define organizational and procedural mechanisms that support the SEPG in their role of identifying potential process improvement areas. For example, these mechanisms can include:

- Assigning a program-level Software Process Engineer (SPE) responsible for facilitating software process tasks for the program.
- Conducting SCM and SQA tasks (see subsections 5.14 and 5.16 and their respective plans)
- Conducting process and product audits by the CSWE and SQA
- Analyzing software management indicators
- Performing the problem reporting and the corrective action process (see subsection 5.17)
- Leading the joint technical and management reviews (see subsection 5.18)

### 5.25.2 Process Audits

The initial audit objective is to verify compliance with the documented procedures. Where non-compliance is found, the cause **must** be identified and evaluated for potential process problems (e.g., lack of training, inadequate documentation, difficulty in implementing the procedure, etc.).

The CSWE, SPE, and SQA (or their designee) **must** attend program and product reviews to witness process compliance as well as to evaluate product quality. Additionally, they should audit segment IPT software tasks for compliance with other software development policies and procedures as defined in the SDP. These audits typically consist of reviewing SDFs, and other software products, and interviewing the software developers performing the particular process being audited. Audit results should be presented to the SEPG.

The SEPG **must** also monitor the monthly software measurement reports to identify problem trends. Where such trends are found, an analysis should be performed to determine if the trend is caused by a process deficiency. An examination of software measurement for the other segments should be performed to determine if similar results are found. Based on these examinations, recommendations for process improvement should be developed and presented to the CSWE for further action.

Proposals for process improvements may also originate from joint technical and management reviews or any segment IPT. In either case, these proposed process improvements **must** be submitted as a Software Change Request (SCR) against the applicable document, or the SDP, and forwarded to the SW/CCB. The SW/CCB can then assign it to the SEPG, or a development site, for evaluation and implementation.

### 5.25.3 Change Implementation

Recommendation for process improvement **must** be presented to the SEPG, for review and evaluation of the cost and schedule impacts, and then to the CSWE for approval. Approved changes at the program level should be implemented by the CSWE. Approved changes concerning a segment SDP Annex should be implemented by the development site. For those actions requiring process training materials, these materials should be generated and used to retrain affected personnel.

Whenever a process change is approved for implementation, the SEPG should provide a recommendation as to the criteria by which they measure the success or failure of the change. Process change implementations should include adequate notification to all affected development and management staff, with a reasonable period of time for resolution of comments, concerns, and questions.

Once the process improvement is implemented, the SEPG should monitor the effect of the change to determine its impact on the software development process. This monitoring not only determines if the desired effect is achieved, but also whether any positive or negative side effects are generated.

### 5.25.4 SEPG Infrastructure

The SDP **must** describe the infrastructure within which the process improvement initiative operates. The segment software Group Lead, or designee, **must** support the SEPGs at the segment and program level. The Program SEPG provides direction to the segment SEPG and may receive services from the corporate SEPG. The Segment IPT Lead is normally a member of the Program SEPG. These relationships, and general functions provided, are shown in the example Figure 5.25.4.

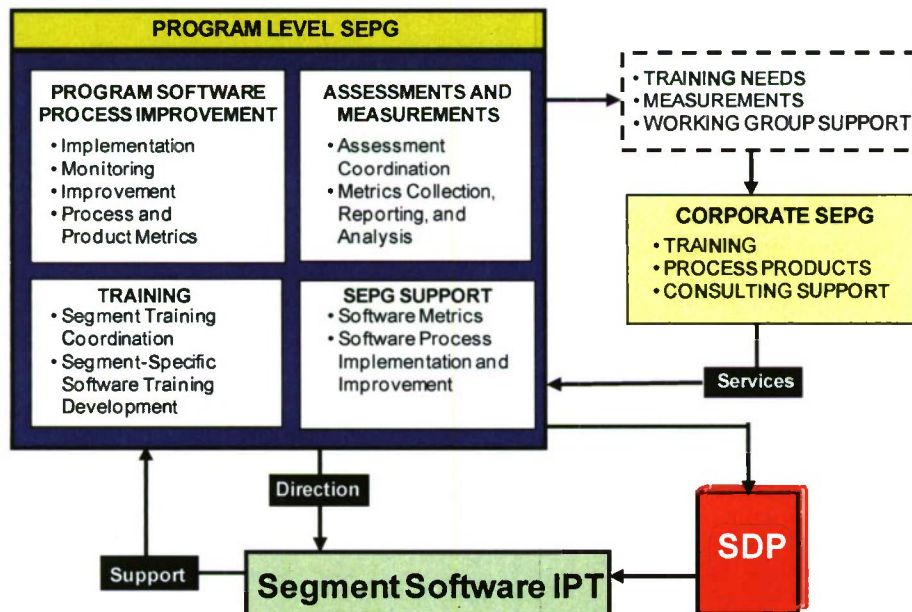


Figure 5.25.4. SEPG Infrastructure—Example

### 5.25.5 Process Training

The cornerstone of an aggressive process improvement program is training. The SEPG (at the segment, program or corporate level) **must** support training in the techniques of process improvement as well as training on the processes.



### 5.25.6 Software Process Engineer/Lead

The Software Process Engineer (SPE), also referred to as the Software Process Lead, is a trained change agent who is responsible for facilitating all software process tasks for the program. The SPE is critical to an effective software process improvement program. The SPE may report to the CSWE, the SEPG Chair, or the software IPT Lead, and usually is the Administrative Chair of the SEPG. Typical functions performed by the SPE are listed in Table 5.25.6.

Table 5.25.6. Typical SPE Functions—Example

Function	Description
SEPG Administrative Chair	Prepare for SEPG meeting (reserve room, agenda, etc.) Prepare and distribute SEPG meeting minutes Ensure that the SEPG follows established applicable program software processes.
Working Group Oversight	Ensure that working groups are effective; ensure that each working group understands roles, responsibilities, and specific task assigned to the working group; monitor progress of working groups, etc.
Software Process Improvement Monitoring and Appraisals	Ensure that the appropriate measurements on process tasks are collected, analyzed, and distributed. Focal point for software process appraisals.
Software Process Improvement Recommendation Coordination	Review, support, and present process improvement recommendations from program personnel, internal and external groups and customer, industry data on current best practices, company and industry standards to the SEPG.
Software Process Improvement Reporting	Report status of SEPG and process improvement activities to software team, program management, company management, and SEPG as applicable.

### 5.26 Software Sustainment (Optional)

There is no requirement in J-16 or TOR 3537B for the inclusion of a subsection covering Software Sustainment (also referred to as Software Support or just as Maintenance). However, it is strongly recommended to include this subsection because Software Sustainment typically consumes about two-thirds (or more) of the total software lifecycle cost. Maintaining a deployed system, responsive to changing customer needs over a long time frame, is as important as the original implementation.

Preparing for software transition to operations and maintenance is discussed in subsections 5.12 and 5.13 of this Guidebook and will not be repeated here. Subsection 5.26 should be focused on the issues an SDP needs to address **during** sustainment and how these issues will be managed and resolved.

#### 5.26.1 Software Sustainment Objectives

The key software development objectives are to produce a software product that provides the required functionality, is easy to access and use, and is cost effective to maintain. The software sustainment tasks needed to satisfy all of these objectives should be identified in the SDP. Generally, there are four types of software sustainment tasks:

- **Corrective:** Fixing errors involves correcting known problems in the software not resolved during development plus correcting problems identified and errors generated after deployment. A critical part of corrective fixes is to update the documentation to facilitate future modifications.
- **Adaptive:** Software must be modified to interface with inevitable changes in the system hardware or operational environment so that the software continues to perform its intended functions. Adaptive maintenance requires identifying the requirements that are affected by the environmental changes, identifying the design changes needed, and implementing those changes in the code.

- **Perfective:** Periodically, the software needs to be enhanced so that it, or the system, accomplishes new functionality. In some cases, this may involve a complete redevelopment effort.
- **Preventive:** Preventive maintenance involves modifications to improve performance, or to implement improvements in maintainability and reliability. During corrective and adaptive maintenance, the users know there is a problem. When modifications are made to make the software “better,” users do not perceive there is a problem so great care must be taken to avoid introducing defects and transforming a good working system into one that does not work well.

Retesting and configuration management play an important part in the process regardless of the type of sustainment performed. Even if the modifications are small, retesting is always required after changes are made to the software. Depending on the extent of the changes, retesting, and regression testing, may be more time consuming, and more complex, than it was for the original system. Retesting must ensure that the original parts still work, that the modifications work as required, and that the functionality of the original system is not inhibited by the changes. The retesting process during sustainment should be documented in detail as this is not a simple task.

Configuration Management (CM), discussed in SDP subsection 5.14, is just as important during sustainment as it was during development. CM manages the changes, determines when the updates are released, and ensures that all users in the field are running the proper version of the software.

## 5.26.2 Planning for Software Sustainment

A Program Office may have three options for planning, addressing, and managing the software sustainment issues typically encountered:

- A Lifecycle Management Plan (LCMP) is a potential option for containing the software sustainment information. However, the LCMP may not be required by the program. If it is required, it may not be produced by the software organization, and typically would not have the software-related details needed. If there is a LCMP it would normally need to be augmented by SDP subsection 5.26.
- A Software Maintenance Plan (SMP) is an excellent mechanism for covering the details needed as described in paragraph 5.26.3. If the SMP is produced by the program, it is recommended that SDP subsection 5.26 contain an overview of it and refer to the SMP for the details.
- If the LCMP or SMP are not produced by the program, or in equivalent documentation, then all of the needed software sustainment details **must** be included in SDP subsection 5.26 or an addendum.

Software Sustainment **must not** be treated as an afterthought. Decisions made during early development activities, especially during software design, can have a significant impact on the cost of sustaining software. DOD systems normally are operational for a long time, so sustainment **must** continue for many years. Planning for software sustainment requires careful consideration in three key activities:

- **Architecture and Design:** Efficient and effective sustainment systems must be well architected and designed. Effective sustainability is a direct result of quality architecture and design. A good design should be modular, highly cohesive (i.e., each module performs a distinct task), and loosely coupled (so that each module depends as little as possible on other modules for its functionality). Design guidelines such as these should be part of, or referenced by, the SDP. The architecture and design must also consider COTS evolution.
- **Coding:** Coding is another important area where improved maintainability can be achieved. Good coding standards are key to this achievement along with an enforcement process.



Frequent peer reviews of the code, during the modifications, also helps to improve maintainability.

- **Documentation:** This is probably the most important activity to improve maintainability. When modifications to the software are needed, software engineers familiar with the system may not be available. The sustainment personnel will have to try to uncover errors and implement fixes based on existing code and documentation. It is vitally important that the software documentation accurately reflects the current state of all parts of the system including the requirements, design rationale, architecture and design models (e.g., UML diagrams), coding, test cases, and test results. Documentation may be hard copy or in electronic format and frequently embedded in tools. If the documentation ever becomes worthless, the inevitable result is a maintenance nightmare.

There are also significant issues that **must** be addressed when upgrading the operational software to new software releases. For example, answers to the following upgrading issues need to be addressed:

- Does the system architecture support isolating a subset of equipment for installing and testing a new software release without affecting ongoing operations?
- Can the same networks be simultaneously used for test data and operational data?
- Can a test database be installed with an operational database?
- Can operations be transferred to an off-site backup facility while the new software release is installed and tested?
- Is the backup facility a full copy and faithful representation of the operational environment?
- Are procedures in place to simultaneously maintain an earlier version of the software while the new version is being developed? In that context, how will changes made in the version under maintenance be incorporated in the new version being developed?

The ability to upgrade current operational systems to new software releases, without affecting ongoing operations, **must** be planned to be incorporated into system requirements from the beginning.

### 5.26.3 Software Maintenance Plan

Adequate facilities, support software, personnel, and documentation **must** be available after the development activities so that the software can be maintained in an operational and sustainable condition. A good way to ensure all of this will be available when needed is to require the contractor to prepare a realistic Software Maintenance Plan (SMP) relatively early during software development. Periodic status reporting should be in place to ensure the plan is followed to accomplish the desired sustainment environment.

Even if the details of the SMP cannot be completed during early development activities, a preliminary draft should be prepared early to ensure the software development process contains the essential sustainment provisions. The SMP should be revised during the development effort—especially after key development milestones. The plan should not be allowed to become out of date or the team responsible for the plan to become dormant. An example outline of the SMP is shown in Table 5.26.3.

Table 5.26.3. Example Outline of the Software Maintenance Plan

Section No.	Title
1.	Introduction (Purpose, goals, and scope of the software maintenance effort)
2.	References (Documents that constrain or support the maintenance effort)
3.	Definitions (Defines or references all terms required to understand the plan)
4.	Software Maintenance Overview (The organization, scheduling priorities, tools, techniques, resources, responsibilities, and methods used in the maintenance process) 4.1 Organization 4.2 Scheduling Priorities 4.3 Resource Summary 4.4 Responsibilities 4.5 Tools, Techniques, and Methods
5.	Software Maintenance Process (Actions performed for each phase of the maintenance process defined in terms of input, output, process, and control) 5.1 Problem/modification identification/classification and prioritization 5.2 Analysis 5.3 Design 5.4 Implementation 5.5 System Testing 5.6 Acceptance Testing 5.7 Delivery 5.8 Risk Management 5.9 Configuration Management
6.	Software Maintenance Reporting Requirements (How information will be collected and provided to members of the maintenance organization)
7.	Software Maintenance Administrative Requirements (The standards, practices, and rules for anomaly resolution and reporting) 7.1 Anomaly Resolution and Reporting 7.2 Deviation Policy 7.3 Control Procedures 7.4 Standards, Practices, and Conventions 7.5 Performance Tracking 7.6 Quality Control of Plan
8.	Software Maintenance Procedures (The procedures to be followed in recording and presenting the outputs of the maintenance process)

#### 5.26.4 The Software Sustainment Organization

The selected software sustainment organization may be a contractor or a government organization. In either case, the selected maintenance organization **must** be able to acquire the knowledge, documentation, support software, and data rights to the software. The software data rights issue should have been addressed before awarding the development contract and should not wait until the sustainment contract.

The Software Sustainment organization **must** have access to the same support software that was used during software development. There will be significant cost savings to the government if the contract specifies that support software, and other related facilities, are included in the contract rather than attempting to acquire them after development is completed. Personnel with the specific software and application knowledge for the system are a key element in establishing the project's maintenance capability. A substantial portion of this knowledge can be retained if key sustainment personnel are involved with reviews, evaluations, and test activities during the software development effort.

#### 5.26.5 Key Software Sustainment Issues

Sustainment becomes more complicated with the increased use of COTS software products. A COTS-intensive system may be a suite of multiple off-the-shelf components—including reuse code, legacy code, and COTS from multiple vendors—plus custom components to achieve desired functionality. Maintaining such a composite of software elements presents a significant software maintenance challenge. COTS software is also discussed in SDP subparagraph 1.2.3.3 and paragraph 4.2.4.



Table 5.26.5 is a summary of key software sustainment issues and pitfalls that are typically encountered; the table pertains to both COTS-intensive systems and systems that are not highly COTS dependent. Subsection 5.26 should address the planned approach for handling these issues and the approach to avoiding the pitfalls typically encountered during the use of COTS software.

Table 5.26.5. Key Software Sustainment Issues

Issue	Resolution
Parallel Testing Capability	Incremental updates and development must take place without affecting ongoing operations.
Planning for Upgrades and Obsolescence	Funding for these upgrades must be planned for. Most COTS software products undergo new releases every two years; old releases will eventually be unsupported.
Software Data Rights	Rights to the source code and documentation is essential; this issue should be resolved before awarding the development contract.
Technology Advancement	The sustainment group must create and maintain a thorough technology refresh plan.
Vendor Licenses	Transition of license management tasks needs to be jointly planned in advance.
Information Assurance Testing	System regression testing must be performed on all upgrades and patches and information assurance requirements must be satisfied.
Design for COTS Interchangeability	Sustainment may be impossible if the COTS vendor goes out of business. The system architecture should isolate the COTS products with minimal interfaces, rather than intermingling COTS with developed software to facilitate replacement. Obtaining the COTS source code (e.g., having it put in escrow) is usually not a good idea as it is almost never adequately documented and is typically so huge that it is impossible to maintain.
Software Risk Management	The sustainment organization must have the resources and capability to identify and analyze risks and perform effective risk mitigation.
Ability to Test Software Updates	Adequate tools, and trained expertise, must be available at the sustainment site and the maintenance environment must support running multiple software versions.
Supporting Processes	Implement an effective and sufficient software fault management process, and other supporting tasks during sustainment.
Qualified Maintenance Personnel	Provide formal training for software maintenance personnel, efficient replacement of key software development staff who leave, and hiring staff members experienced in managing COTS-intensive systems.
Incomplete Software Sustainment Documentation	Keep software documentation up-to-date and usable. Specify the delivery of a complete set of software documentation, in sufficient detail, in the development contract. Documentation includes text, code, models, diagrams, etc.
Ineffective Configuration Management	An effective Configuration Management process is essential to a successful sustainment effort. The CM system must be able to handle multiple versions in support concurrently and must maintain the exact version and configuration of the software at each operational site, and in each computer at the operational sites. That means the CM system must track the software version installed in each workstation and server. This is especially important during upgrades.
Version Synchronization	Maintain synchronization of development versions by feeding back into the development versions the changes made to the version being supported. If this is not performed effectively, the result will be multiple diverging baselines of the same system with a large cost downstream to get everything in synchronization.





## 6. Schedules and Activity Network

Every major program **must** have and follow a formal program management methodology by maintaining an approved Integrated Master Plan (IMP) coupled with an Integrated Master Schedule (IMS), or equivalent, to provide a complete schedule and activity network for all program activities. The IMP and IMS **must** be maintained electronically and available through an electronic data management system.

**IMP/IMS/CWBS.** The IMP and IMS **must** be organized by a systematic Contract Work Breakdown Structure (CWBS) to provide a complete schedule and activity network for all program activities. The IMS **must** include software activities showing the time-phased interrelationships of events and accomplishments for software builds, and the IPTs **must** manage and control their respective schedules within the IMS structure.

If the IMS is at a relatively high level, it **must** be augmented with lower-level detailed segment schedules for software planning, design, development, integration, and test.

These detailed segment schedules **must** be maintained and monitored at the segment level with oversight by Program Management and the CSWE. Segment schedules **must** be integrated with the IMS. If any conflicts between the IMS and segment schedules occur, the IMS always prevails.

**Software Schedules.** The summary and detailed schedules for the software activities can be updated weekly (or monthly) to be consistent with overall program schedules. The software development schedules **must** show the details of the proposed builds and how they relate to overall program milestones. Eventually, the software schedules should get all the way down to the “inch stones” with tasks identified at the level of individual engineers. Typically, the schedules are prepared by “rolling waves” which can be for a six month period or build by build.

To properly account for software related costs, CWBS elements **must** be created that allow software costs to be properly assigned to the correct categories. IPT leads and IPT software leads should status the schedule, perform analysis and trending, identify problem areas, develop action plans, and brief IPT management.

An overall master schedule may be included in Section 6 with the SDP submitted with the proposal. However, once the contract starts, the schedules, especially the detailed software schedules, are typically updated so frequently that they should only be referenced in SDP Section 6.

**Software Activity Network.** The SDP should include, or reference, an activity network depicting sequential relationships and dependencies among software activities, and identify those activities that impose the greatest restrictions on the project. The activity network identifies the critical path. If any part of the software development is on the segment or program critical path, then additional management attention is needed to address the issue.





## 7. Project Organization and Resources

This section of the SDP **must** describe the organizational structure to be used on the project, including the contractor team members involved, their relationships to one another, the authority and responsibility of each organization for carrying out required activities, and the resources to be applied to the project. In accordance with TOR-3537B, there **must** be two subsections:

- 7.1 Project Organization
- 7.2 Project Resources

### 7.1 Project Organization

An overview of the program organization structure **must** be provided in this subsection. A top level view of the software organization, and an overview of the Software Items, was recommended earlier to be included in SDP subsection 1.1 (see Figure 1.1). Since subsection 7.1 is focused on the “project” organization, it **must** contain a project organization chart showing the relationship of the software organizations to the overall program. This subsection can include references to the software team responsibilities table (see Table 3.3) and the build delivery plan table (see Table 5.1.1.3). Figure 7.1 is an example of an overall organization chart with an emphasis on the software elements. The program organization should facilitate software management visibility and software technical oversight.

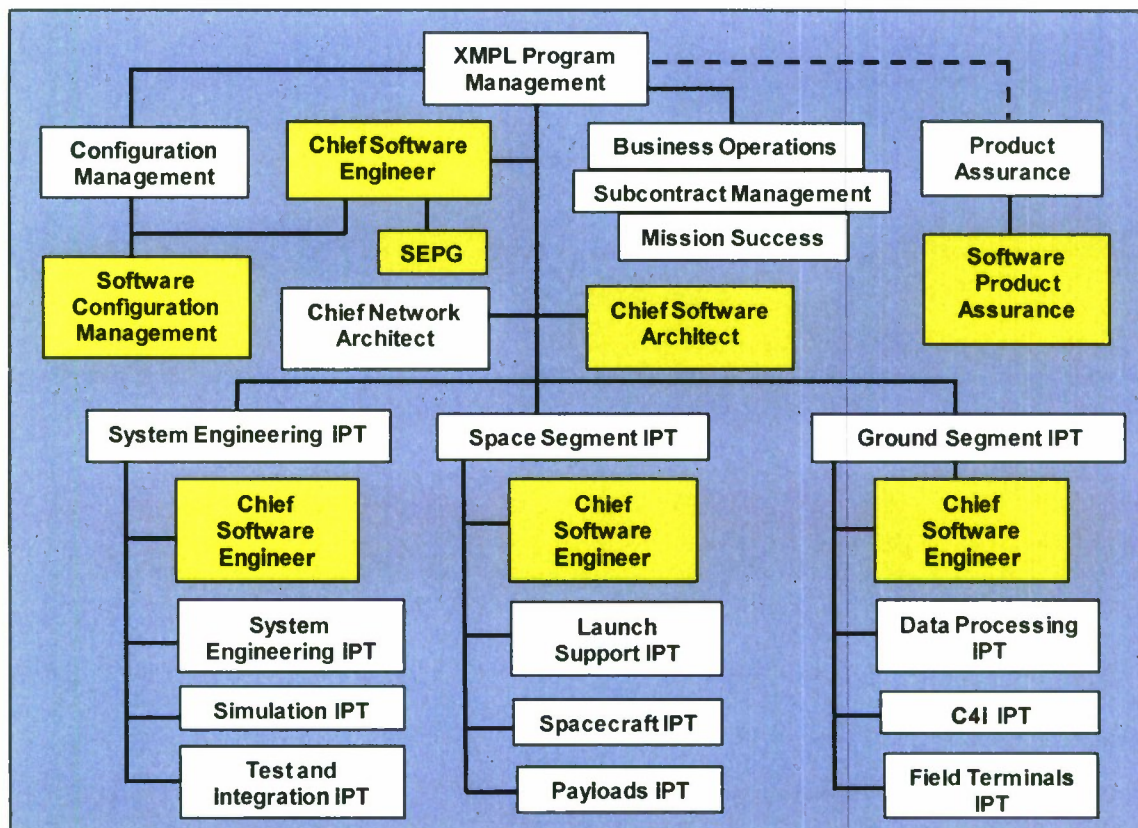


Figure 7.1. Overall Program Organization—Example

The overall organization, responsibilities, and management approach **must** be described in the Integrated Master Plan (IMP). The Integrated Management Schedule (IMS) **must** provide program level schedules, timelines and required resources. The program management and Integrated Process

and Product Development (IPPD) process **must** establish a clear structure and unambiguous responsibility for program participants. The Contract Work Breakdown Structure (CWBS) **must** subdivide the program into clearly defined, tracked and manageable discrete tasks. **Contractual work assignments, based on the IMP, IMS, IPPD, and WBS are the basis for the formal relationship among the customer, the prime contractor, and the subcontractors.** Each contractor should be responsible for software at the software item (SI) level.

**Integrated Product Teams.** The most effective organizations consist of a hierarchy of Integrated Product Teams (IPT). The IMP should identify responsibilities for the IPTs. The contract should identify key personnel and their responsibilities during software planning, development, test, and deployment. The IMP should designate software responsibilities through identification of program events, planned accomplishments, and acceptance criteria for the accomplishment.

IPTs are the basic performing organization. They have resources and authority to deliver products and execute processes. The IPT is essentially a matrix organization, assembled from members of the applicable engineering disciplines to manage the design, development, production, and support of a product system. The team organization is normally consistent with the product hierarchy as defined in the CWBS, with each CWBS element the primary responsibility of a single IPT. This allows IPTs to identify clear and measurable outputs and necessary interfaces and it facilitates the flowdown of requirements to the IPTs.

**Software Engineering Process Group (SEPG).** Software development activities span organizational, administrative, geographic, and functional boundaries. The SEPG is an important organizational element as it ensures consistent implementation of the software development processes and production of compatible software products. Responsibilities include planning, managing, and/or coordinating with: Software Configuration Management (SCM); Software Quality Assurance (SQA); corporate SEPG support; software training; the CCB and SW/CCB; the IPTs to improve development processes and training; and coordinating, developing, and maintaining internal software work instructions and procedures (see subsection 5.25 for details on SEPG functions).

## 7.2 Project Resources

The resources that **must** be covered in this section cover: key personnel, including staff-loading, skill levels and responsibilities; developer facilities including geographic locations of team members; acquirer-furnished equipment; and other required resources needed for the program.

### 7.2.1 Personnel Resources

**Documentation of the software organization must include key supporting roles,** including the Chief Software Engineer (CSWE), Software Process Engineer/Lead, Software Configuration Management (SCM) Lead, Software Quality Assurance (SQA) Lead; IPT software leads, Software Integration and Test Lead, SI Leads, Software Engineers, Software Subcontract Management, and data management. Roles and responsibilities for the major development activities were discussed in SDP subsections 5.5 through 5.9. In addition, the tables in Appendix A of this Guidebook contain example summaries of roles and responsibilities for the key software engineering skill groups.

#### 7.2.1.1 Chief Software Engineer

**A key element of success, for a software-intensive program, is the establishment of an effective and proactive CSWE team. It is generally understood that the quality of a software product is directly related to the process used to create it—and the CSWE is the core of the process.**



The CSWE should be accountable to, and report directly to the program manager, or to the System Engineering manager, and is the primary point of contact with the Program Office for all software matters across the program. (In some programs, this responsibility may be shared with a Chief Software Architect). The CSWE should be a voting member of the appropriate control boards and the risk management board. Table 7.2.1 is an example list of responsibilities the CSWE team typically has for software oversight as well as process guidance.

Table 7.2.1. Chief Software Engineer Team Responsibilities—Example

Chief Software Engineer – Software Development Oversight Responsibilities	
Reviewing the definition of software elements, or subsystems needed to satisfy requirements	
Reviewing of high level software architecture guidance to be used in development of software implementing details through design reviews and technical exchange meetings	
Reviewing of software baseline implementation and subsequent changes	
Assessing and guiding the IPTs/teammates SCM requirements, build approach, and implementation	
Assessing the methods used to transition new builds into the operational baseline	
Selecting program tools, developing program-level training	
Oversight of architecture development and data architecture, including the database management system, file system implementations, and support tools to ensure successful execution	
Assessment validation and verification approaches and segment operability	
Reviewing segment timeline performance analysis	
In concert with the SEPG, collecting and consolidating metrics from the IPTs	
Analyzing metrics across the program and providing metric summaries and recommendations	
Reviewing and approving software plans, specifications, test procedures, and test results	
Definition and oversight of software IV&V activities	
Technology insertion planning and review	
Chief Software Engineer – Software Process Oversight Responsibilities	
Developing the Program-wide SDP, and monitoring for compliance by the IPTs/teammates	
Reviewing software implementation processes to ensure compatibility with goals of high quality, cost effective architecture development	
Defining and implementing the program's Quantitative Management Plan	
Assessing cost, schedule, technical and management risk of all software IPTs/teammates	
Sustaining all program level common software tools	
Chairing the SEPG and the Software Configuration Control Board (SW/CCB)	
Chief Software Engineer – Software Support to SEIT	
Participating, assessing, and approving requirements allocations and decompositions to software elements, domains, or subsystems	
Assisting the SEIT in trade studies and margin assessments related to software	
Reviewing and approving subcontracts requiring software development or purchase of software	

### 7.2.1.2 Staff Loading

The required time-phasing of software development personnel is normally documented in the program's Earned Value Management System (EVMS) database. EVMS is the basis for monthly software cost/schedule reporting and tracking for each WBS element. Software staffing varies during the program from an initial build-up to a peak and then a gradual decline as the majority of the software effort is completed. **An estimated staff-loading chart must be included in this subparagraph of the SDP** and Figure 7.2.1.2 is one example of how it may be depicted.

It is the responsibility of the acquisition team, as well as the contractor, to analyze the software planning to ensure its executability. The size of the software task, the schedules in place, and the planned staff loading must form an executable software development effort across all the software team members.

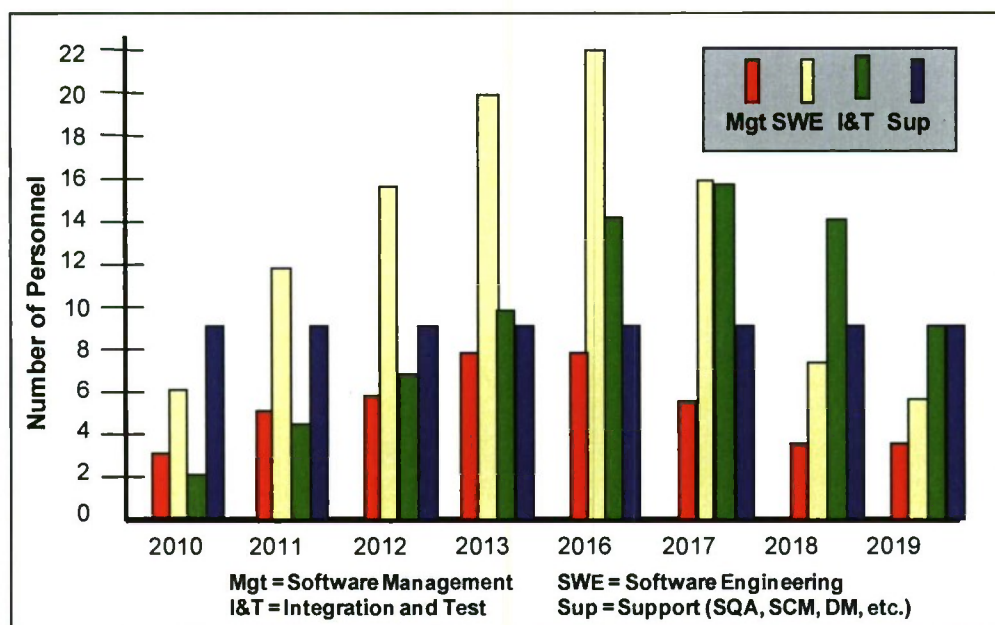


Figure 7.2.1.2. Estimated Software Staff-Loading—Example

### 7.2.1.3 Skill Levels

A breakdown of the planned skill levels, geographic locations of the skills needed, and the security clearances of personnel performing the work are all required by TOR-3537B and J-16. The software leads typically estimate and maintain the staffing profiles and work effort loading distributions. This data is reported in the Cost Analysis Requirements Description (CARD) document. Table 7.2.1.3 is a tabular example of how to show the skill level requirements, by company and by software function. Security-related information should also be discussed.

Table 7.2.1.3. Estimated Skill Levels By Location and Function—Example

Able Corporation							
Anytown, CA		Skill Level					
	Eng 1	Eng 2	Eng 3	Tec 1	Tec 2	Tec 3	Total
Software Engineering	0	10	12	2.5	1.5	0	29
Software Quality Assurance	1	1	2	0	0	.2	4.2
Software Configuration Mgt	1	2	2	0	.5	0	5.5
Software Testing	1.5	1.4	1.1	.4	.2	.2	4.8
Software Management	2	2	.5	0	0	0	4.5
Total Staff:		8.5	16.4	17.6	2.9	2.2	.4
Skill Level		Skill Level Definitions					
Engineer 1	Senior Level with 13 or more years experience and...						
Engineer 2	Mid-Level with 3–12 years experience and...						
Engineer 3	Entry level with 0–2 years experience and...						
Technician 1	An expert in their discipline with 13 or more years experience and...						
Technician 2	Emerging authority with 8 or more years experience and...						
Technician 3	Entry level to experienced with an advanced degree and...						



## 7.2.2 Development Facilities

An overview of developer facilities to be used, including geographic locations in which the work will be performed, facilities to be used, secure areas, and special features of the facilities applicable to the contract are all required by TOR-3537B. In addition, software development activities taking place at each location should be identified. Table 7.2.2-1 is an example of a simple table showing the team members, their locations, and their corresponding software responsibilities.

Table 7.2.2-1. Team Locations and Software Activities—Example

Company	Location	Software Development Activities
Able Corp.	Anytown, CA	Program Management; Mission Planning; Test and Integration
Baker Co.	Somewhere, NJ	Payload Management; Ground Systems; Common Services
Charlie Co.	Metropolis, MD	Network Management; Operations Management
Delta Corp.	Goodtown, PA	Security Management; Resource Allocation
Epsilon Co.	Smalltown, CO	Spacecraft Control; Satellite Network
Gamma Co.	Someplace, AZ	Operations Management; Data Services

A companion table should be used to summarize the facilities allocated to the program at each team location for software development. Table 7.2.2-2 is an example of a simple table that is a breakdown of the square feet for office space and lab space allocated for software development. The Software Engineering Environment (SEE) was covered in SDP subsection 5.2 and Table 5.2.1-2 covered the major facilities, however, Table 7.2.2-2 should have more detail regarding facilities. There should be a clear mapping between the SEE in SDP subsection 5.2 and the development facilities identified in paragraphs 7.2.2, 7.2.3, and 7.2.4. In addition, the development environment hardware and tools should be mapped to each facility location.

Table 7.2.2-2. Facilities Allocation—Example

Company	Location	Total Sq. Ft.	Assigned Sq. Ft.	Office Sq. Ft.	Lab Sq. Ft.
Able Corp.	Anytown, CA	500,000	150,000	120,000	30,000
Baker Co.	Somewhere, NJ	450,000	50,000	10,000	10,000
Charlie Co.	Metropolis, MD	65,000	15,000	10,000	5,000
Delta Corp.	Goodtown, PA	80,000	20,000	10,000	5,000
Epsilon Co.	Smalltown, CO	10,000	6,000	5,000	1,000
Gamma Co.	Someplace, AZ	12,000	3,000	2,500	500
Total Square Feet		1,117,000	244,000	192,500	51,500

## 7.2.3 Government Furnished Equipment, Software and Services

This paragraph **must** contain an identification of Government Furnished Equipment (GFE), software services, documentation, data and facilities required for the contract effort. In most cases, the details are not contained in the SDP but coordinated and documented by contracts and listed in a separate document. A GFE summarization table can be included in the SDP. Additions and deletions to the GFE list should be reviewed and approved by program management and key technical personnel prior to approval.

## 7.2.4 Other Required Resources

In addition to personnel resources, facilities and GFE, the timely definition and deployment of key physical resources are required to successfully execute the program. The Master Facilities Plan

should define facilities requirements for the program. Examples of key physical resources that may need to be developed and provided by the contracting team(s) are:

- Communication capabilities for the secure exchange of sensitive and classified information among the contracting teams and the government
- Contractor software development facilities including desktop computers, networks, and the software engineering environment (SEE) including tools
- Mission Simulation System facilities for the development of simulation software
- A flight and payload software vehicle simulator test bed for high-fidelity simulations of on-orbit satellites
- Spacecraft bus and payload Software Development Facility for end-to-end flight software testing
- Backup storage facilities for disaster recovery

### 7.2.5 Software Training Plans (Optional)

This paragraph is not required by TOR-3537B or J-16 but is highly recommended since training resources, specifically the funds allocated to provide an adequate training program, is often underfunded or even neglected. **Program Training Plans must be developed to address software training needs.** The training plans should be developed, maintained, and monitored by training coordinators and/or software process leads in coordination with IPT software leads.

The plans should address program specific technical and process training, identify training requirements by job category, provide for a waiver procedure, and require training records to track completion. Training plans should go all the way down to individuals, including the training they have already had, what training they still need to take, and when they are scheduled to take it. The SDP, or a separate training plan if one is produced, should make a clear distinction between:

- Basic training provided by the contracting organization that is funded by that organization
- Program-specific training provided under contract funding

Contractors are responsible for staffing the program with qualified people and for providing program specific technical and process training. **Each organization must develop coordinated plans to implement training in accordance with its organizational practices.** Program-specific training should include:

- An introduction for new employees on the program
- Technical and management oversight
- A summary of the processes and methodologies used
- Where information may be obtained
- The instruction required for the efficient use of COTS products and development tools

The IPT leads and IPT Software Leads, assisted by their respective functional staffing manager, should provide training guidance to their staff. This may include suggestions for either technical enhancement or career development training. Periodic lists of upcoming training classes should be provided by the training coordinators to program personnel. When necessary, the program should request training from the training organization to achieve specific training requirements.



## 8. Notes

Section 8 of the SDP must include general information that aids in the understanding of the SDP or long tables that may interfere with efficient reading of the SDP. To be compliant with TOR-3537B, Section 8 must include all acronyms used in the SDP, including abbreviations and what they mean, plus a list of definitions or terms used. The tables below show examples of:

- Software Acronyms used in the SDP (Table 8.1)
- Software-Related Definitions to clarify meanings of the terms used in the SDP (Table 8.2)
- A list of work instructions and procedures, external to the SDP, that defines “how” to do it as opposed to the SDP that is focused on “what” is done and who does it (Table 8.3)

Table 8.1. Acronyms—Example

Acronym	Definition
API	Application Programming Interface
APO	Acquisition Program Office
BAR	Build Architecture Review
BOE	Basis of Estimate
BTR	Build Turnover Review (or PTR or TER)
CAP	Corrective Action Plan
CAIV	Cost As An Independent Variable
CARD	Cost Analysis Requirements Document
CASE	Computer Aided Software Engineering
CBA-IPi	CMM <sup>SM</sup> -Based Assessment – Internal Process Improvement
CCB	Configuration Control Board
CCR	Critical Computer Resource
CDD	Capabilities Development Document
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CIP	Contract Implementation Plan
CM	Configuration Management
CMMi <sup>SM</sup>	Capability Maturity Model – Integrated
CMP	Configuration Management Plan
COM	Computer Operation Manual
	Etc.

Table 8.2. Software-Related Definitions—Example

Glossary	Definition
Algorithm Software	Operational Algorithm Code – Algorithm code that has been verified to meet all functional and performance requirements for data quality, timeliness, and execution within the architecture.
Algorithm Team	Interdisciplinary team of scientific and engineering personnel assigned to the verification, development, and testing of a specific set of algorithms. Responsibilities include technical resolution of data quality or timeliness requirements issues.
Baselines	Software baselines describe a particular version of software (e.g., increment, build, or release) and consists of a set of internally consistent requirements, design, code, build files, and user documentation. A requirements baseline includes an SRS under CCB control.
Delivery	The process of providing a software product that is ready for acceptance by a higher-tier organization for the purpose of fulfilling a contractual requirement.
	Etc.

Table 8.3 is an example list of Work Instructions and Operational Procedures that may be used by developers while implementing the processes defined in the SDP. These are “how to” instructions. This list should be provided in the SDP as a reference, however, the SDP generally requires government access to all the Work Instructions and Operational Procedures to be used by the software contractors and their subcontractors.

Table 8.3. Work Instructions and Procedures—Example

Work Instruction Number	Work Instruction Name
#####	Action Item Processing
#####	Build Architecture, Planning and Review
	Change Control Process
	Code and Unit Test Planning and Reporting
	Configuration Management—Audits and Reports
	COTS Baseline Management
	COTS Product Evaluations
	Critical Design Reviews
	Delivery of SDRL Items
	Delphi Estimation Technique
	Deviations and Waivers
	Disaster/Backup Storage
	Electronic Data Management
	External Communication
<b>EXAMPLE</b>	Formal Reviews
	Integration Readiness Review
<b>LIST OF</b>	Recording Meeting Minutes
	Requirements Review and Specification Generation
<b>WORK INSTRUCTIONS</b>	Schedule Development, Approval, and Maintenance
	Software Build Release Review
<b>AND</b>	Software Configuration Control Board
	Software Design
<b>OPERATIONAL</b>	Software Development Folder (SDF)
	Software Disaster Recovery
<b>PROCEDURES</b>	Software Document Review and Approval
	Software Engineering Notebook (SWEN)
	Software Integration and Checkout
	Software Integration and Test
	Software Measurements
	Software Peer Reviews
	Software Planning
	Software Process Documentation
	Software Status and Progress Reporting
	Software Requirements
	Software Size Effort Estimation
	Software Test Cases
	Software Test Procedures
	Software Test Reports
	Software Unit /Component Test
	SQA Process Evaluation and Quality Records
	SQA Product Evaluation
	SQA Training
	Test Tool Validation
	Etc.



### **Part 3. Additional SDP Guidebook Information**

The following Additional Guidebook Information (AGI) to this SDP Guidebook is not intended to be an integral part of an SDP. The recommended contents of a program-level SDP are described in Part 2 of this Guidebook. The purpose of the following additional information is to make the SDP Guidebook more useable and to support the principal objectives of the Guidebook including assisting acquisition agencies in evaluating SDPs, providing guidance to contractors in preparing and updating their SDPs, and describing the systematic steps of the software development process lifecycle.

- AGI-1: Software Roles and Responsibilities
- AGI-2: Bibliography
- AGI-3: Software Related Definitions
- AGI-4: Software Acronyms
- AGI-5: Subject Index to the SDP Guidebook





## **AGI-1. Software Roles and Responsibilities**

Tables AGI-1 through AGI-11, summarize the roles and responsibilities for the following software engineering skill groups:

- Chief Software Engineer (Table AGI-1)
- Segment Chief Software Engineer (Table AGI-2)
- Software Process Lead (Table AGI-3)
- IPT Software Lead (Table AGI-4)
- IPT Software Integration and Test Lead (Table AGI-5)
- Software Item Lead (Table AGI-6)
- Software Engineer (Table AGI-7)
- Software Test Engineer (Table AGI-8)
- Software Configuration Management (Table AGI-9)
- Software Quality Assurance Management (Table AGI-10)
- Software Subcontract Management (Table AGI-11)

The information in these tables is presented as an example of software roles and responsibilities—they are not intended to limit an individual's responsibilities. The intent is to define a minimum set of responsibilities and how various individuals interact to facilitate consistency across a program. Tables like these could be included in, or referenced by, the various SDP sections, subsections, paragraphs, and subparagraphs where the listed software engineering roles and responsibilities are performed. SDP subsections 5.5 through 5.9, the core of the software development process, also contains summary lists of roles and responsibilities.

Table AGI-1. Roles and Responsibilities of the Chief Software Engineer—Example

<b>Roles</b>	<b>Responsibility</b>
<b>Chief Software Engineer</b>	Provide software oversight and insight across the program Lead and coordinate system software activities Report overall software status to program management Empowered to work software problems at any level Run and prepare (if applicable) Software Management Reviews Recommend award fee score for software subcontractors
<b>SEPG</b>	SEPG Chair
<b>SDP</b>	Key contributor, reviewer, and approver
<b>Software Appraisal</b>	Prepare for and support appraisal
<b>Work Instructions and Procedures</b>	Review all work instructions, procedures, and local processes
<b>Planning</b>	System level software planning
	Review all software schedules for consistency across program
<b>Cost/Schedule Reporting</b>	Review software IPT information
<b>Requirements</b>	Review and assess software requirements across the program
<b>Architecture Design</b>	Review/assess software architecture design across the program Contributor, reviewer, and approver of system architecture documents
<b>Risk Management</b>	Represents software at the Risk Management Board
<b>Software Test</b>	Review and assess software test plans, procedures, and reports across the program
<b>Software Metrics</b>	Consolidate metrics from IPTs into program level metrics Analyze metrics for trends across program Facilitate program and IPT level action in response to metrics Coordinate definitions of metrics and measurements
<b>Problem Reports</b>	Address problems that span IPTs and system interfaces
	Address problems that affect functional performance
<b>Review Boards</b>	Engineering Review Board (ERB) Segment IPT Software Configuration Control Board as needed Program Configuration Control Board (CCB)

Also see SDP subparagraph 7.2.1.1 for a description of responsibilities for the Chief Software Engineer's team.



Table AGI-2. Roles and Responsibilities of the Segment Chief Software Engineer—Example

<b>Roles</b>	<b>Responsibility</b>
<b>Segments Chief Software Engineer</b>	Provide oversight and insight into the ground segments or space segment (as applicable) software activities Lead and coordinate the segment's software activities Report the segment's software status to chief software engineer and program management Empowered to work software problems at any level within the team Prepare for and support Software Management Reviews
<b>SEPG</b>	Member
<b>SDP</b>	Key contributor and reviewer
<b>Software Appraisal</b>	Prepare for and support appraisal
<b>Work Instructions and Procedures</b>	Review work instructions, procedures, and local processes
<b>Planning</b>	Perform segment level software planning Review software schedules for consistency across program
<b>Cost/Schedule Reporting</b>	Review the segment's IPT information
<b>Requirements</b>	Review and assess software requirements across the segment
<b>Architecture Design</b>	Contribute, review, and assess software architecture/design across the segment
<b>Risk Management</b>	Represents segment software at the Risk Management Board as required Identify, assess, mitigate, monitor, and close software risks
<b>Software Test</b>	Review and assess software test plans, procedures, and reports across the segment
<b>Software Metrics</b>	Consolidate metrics from the segment's IPTs into program level metrics Analyze metrics for trends across the segment's software items Facilitate program and IPT level action in response to metrics Coordinate metric definition
<b>Problem Reports</b>	Address problems that span the segment's IPTs Address problems that span system interfaces Address problems that affect functional performance
<b>Review Boards</b>	Lead the segment's IPT SCCBs Member of the program SCCB

Table AGI-3. Roles and Responsibilities of the Software Process Lead—Example

<b>Roles</b>	<b>Responsibility</b>
<b>Software Process Lead or Software Process Engineer</b>	Software process owner; responsible for defining and maintaining the program's software process captured in the SDP
	Represent the program with corporate SEPG
	Plan and coordinate software training
<b>SEPG</b>	SEPG Administrative Chair Chair forum for review and concurrence of software work instructions/ procedures Chair horizontal coordination of software work instructions/procedures across IPTs
<b>SDP</b>	SDP owner
<b>Software Appraisal</b>	Lead software appraisal preparation effort and customer interface (customer may perform the assessments)
<b>Work Instructions and Procedures</b>	Plan and coordinate procedure development
<b>Planning</b>	Owner of program-common software procedures Review and concur with work instructions/procedures that implement the processes called out in the SDP
<b>Cost/Schedule Reporting</b>	Content and format Review software process cost schedule tracking
<b>Requirements</b>	Reviewer
<b>Architecture Design</b>	Reviewer
<b>Risk Management</b>	Maintains the program software risk mitigation plan
<b>Software Test</b>	Reviewer
<b>Software Metrics</b>	Content and format Reviewer
<b>Problem Reports</b>	Reviewer Elevate problem trends to corporate SEPG
<b>Training</b>	Ensure development and maintenance of Program Training Plan



Table AGI-4. Roles and Responsibilities of the IPT Software Lead—Example

<b>Roles</b>	<b>Responsibility</b>
<b>IPT Software Lead</b>	Lead and coordinate segment software activities
	Report IPT software status to IPT leader
	Advise, coach, and resolve conflicts within the IPT software team
	Prepare for and support Software Management Reviews
<b>SEPG</b>	Member
<b>SDP</b>	Key contributor, reviewer, and stakeholder
<b>Software Appraisal</b>	Prepare for and support appraisal
<b>Work Instructions and Procedures</b>	Ensure work instructions/procedures are followed
<b>Planning</b>	IPT level software planning
	Responsible for segment software bidding information
<b>Cost/Schedule Reporting</b>	Manage IPT software baselined schedule
	Manage IPT software budget (if not allocated to the SI level)
	Capture earned value (if budget held at this level)
	Consolidate SI earned value into IPT software earned value
	Note: budget must be portioned to SIs and earned value collected at the SI level
<b>Requirements</b>	Support allocation of requirements to SIs
<b>Architecture Design</b>	Provide technical guidance on trade studies, systems engineering, design, and vendor selection
<b>Risk Management</b>	Identify, assess, mitigate, monitor, and close software risks
<b>Software Test</b>	Provide technical guidance on integration and testing
<b>Software Metrics</b>	Consolidate metrics from SIs into IPT level metrics
	Collect IPT level metrics
	Analyze metrics for trends across IPT
	Facilitate IPT level action in response to metrics
<b>Problem Reports</b>	Track software problems and ensure problems are resolved by due date
<b>Review Boards</b>	Segment IPT SCCB chair

Table AGI-5. Roles and Responsibilities of the IPT Software Integration and Test Lead—Example

<b>Roles</b>	<b>Responsibility</b>
<b>IPT Software Integration and Test Lead</b>	Ensure software meets the requirements defined in the SRSs Ensure IPT SIs collectively meet the segment specification (as appropriate) Perform SI integration
<b>SEPG</b>	Member
<b>SDP</b>	Reviewer
<b>Software Appraisal</b>	Prepare for and support appraisal
<b>Work Instructions and Procedures</b>	Prepare test specific work instructions/procedures
<b>Planning</b>	IPT level software integration and test planning
<b>Cost/Schedule Reporting</b>	SI earned value review; IPT software earned value, cost, schedule
<b>Risk Management</b>	Identify, assess, mitigate, monitor, and close software risks
<b>Software Test</b>	Coordinate integration and testing between software teams
	Owner of segment software integration sequence, integration test plans, threads, cases, and procedures
	Integration test lead
	Review and approve integration test results
<b>Software Metrics</b>	Collect and report software integration and test metrics
	Analyze metrics and take appropriate action
<b>Problem Reports</b>	Record and track problems
<b>Review Boards</b>	Segment IPT SCCB member



Table AGI-6. Roles and Responsibilities of the Software Item Lead—Example

<b>Roles</b>	<b>Responsibility</b>
<b>Software Item Lead</b>	Lead and coordinate the SI software activities
<b>SEPG</b>	Member
<b>SDP</b>	Reviewer
<b>Software Appraisal</b>	Prepare for and support appraisal
<b>Work Instructions and Procedures</b>	Implement work instructions/procedures
<b>Planning</b>	Perform SI level software planning
<b>Cost/Schedule Reporting</b>	Manage SI software team based on baselined schedule Manage SI budget (if not held by the IPT software lead) Capture earned value (if budget held at this level) Report schedule status to IPT software lead
<b>Requirements</b>	Allocate requirements to lower level and higher level SUs Assign SUs to software engineers
<b>Architecture Design</b>	Oversee SI-level architecture design Coordinate inter-SI interface design Verify SU design meets SI requirements
<b>Risk Management</b>	Identify, assess, mitigate, monitor, and close software risks
<b>Software Test</b>	Review SU testing for SI integration Develop SI test plans/procedures for SI integration
<b>Software Metrics</b>	Collect and report SI metrics Analyze metrics and takes appropriate action
<b>Problem Reports</b>	Assign problems to software engineers

Table AGI-7. Roles and Responsibilities of the Software Engineer—Example

<b>Roles</b>	<b>Responsibility</b>
<b>Software Engineer</b>	Development and test of individual SUs
<b>SEPG</b>	Participate as needed
<b>SDP</b>	Understand and follow
<b>Software Appraisal</b>	Prepare for and support appraisal as needed
<b>Work Instructions and Procedures</b>	Implement work instructions/procedures
<b>Planning</b>	SU level software planning
<b>Cost/Schedule Reporting</b>	Schedule/report SU activities
<b>Requirements</b>	Define/derive SI requirements and interfaces for assigned tasks
<b>Architecture Design</b>	Design and develop assigned SU architecture
<b>Risk Management</b>	Identify, assess, mitigate, monitor, and help close software risks
<b>Software Test</b>	Code and unit tests assigned SUs Perform SI integration and test
<b>Software Metrics</b>	Collect SU level information and provides to SI lead
<b>Problem Reports</b>	Work assigned problems



Table AGI-8. Roles and Responsibilities of the Software Test Engineer—Example

<b>Roles</b>	<b>Responsibility</b>
<b>Software Test Engineer</b>	Perform verification of software requirements
<b>SEPG</b>	Participate as needed
<b>SDP</b>	Understand and follow
<b>Software Appraisal</b>	Support appraisal as needed
<b>Work Instructions and Procedures</b>	Implement work instructions/procedures
<b>Planning</b>	SI qualification test planning
<b>Cost/Schedule Reporting</b>	Schedule/report SI qualification test activities
<b>Requirements</b>	Test software to meet requirements
<b>Risk Management</b>	Identify, assess, mitigate, monitor, and help close software risks
<b>Software Test</b>	Develop, dry run and execute test procedures, test cases, test data, databases, test drivers, scripts, etc., and report results
<b>Software Metrics</b>	Collect SI qualification test information and provide to SI test lead
<b>Problem Reports</b>	Work assigned problems

Table AGI-9. Roles and Responsibilities of the Software Configuration Management—Example

Roles	Responsibility
<b>Software Configuration Management</b>	Establish software baselines; identify items to be placed under software CM control; manage changes to items under software CM control
	Perform baseline status accounting
	Perform subcontractor software baseline library audits Manage COTS software and changes to COTS software in the development environment and in operational software
<b>SEPG</b>	Member
<b>SDP</b>	Contributor, reviewer, and approver
<b>Software Appraisal</b>	Prepare for and support appraisal
<b>Work Instructions and Procedures</b>	Author SCM specific work instructions/procedures
<b>Requirements</b>	Establish requirements baseline
<b>Architecture Design</b>	Establish architecture design baselines
<b>Software Test</b>	Build test software from source code and provide configured software for testing
<b>Software Metrics</b>	Collect and report problem report metrics
<b>Problem Reports</b>	Track problems
<b>Review Boards</b>	Administer local Segment IPT SCCB



Table AGI-10. Roles and Responsibilities of the Software Quality Assurance Management—Example

Roles	Responsibility
<b>Software Quality Assurance Management</b>	Monitor compliance with program and corporate processes and standards
	Report to the program Product Assurance manager
	Report status and findings to SI lead, IPT software lead, chief software engineer, software process lead, IPT leads, and program manager
	Perform subcontractor software quality assurance system audits
<b>SEPG</b>	Member
<b>SDP</b>	Contributor and reviewer; SQA Lead is an approver
<b>Software Appraisal</b>	Prepare for and support appraisal
<b>Work Instructions and Procedures</b>	Author SQA specific work instructions/procedures
<b>Requirements</b>	Audits requirement baseline
<b>Architecture Design</b>	Audit design baseline
<b>Risk Management</b>	Identify, assess, mitigate, monitor, and help close software risks
<b>Software Test</b>	Audit configured software
	Witness testing where requirements are verified
<b>Software Metrics</b>	Collect and report audit metrics and SQA non-compliance metrics
<b>Problem Reports</b>	Close the SQA non-compliance reports
<b>Review Boards</b>	Segment IPT SCCB member

Table AGI-11. Roles and Responsibilities of the Software Subcontract Management—Example

Roles	Responsibility
<b>Software Subcontract Management</b>	Technical subcontract aspects of a software subcontract Function similar to an IPT software lead over the subcontract scope
	Participate in peer reviews
	Perform software subcontractor oversight
	Approve all SDRLs as called out in the contract
<b>SEPG</b>	Member
<b>SDP</b>	Contributor, reviewer, and stakeholder
<b>Software Appraisal</b>	Prepare for and support appraisal
<b>Work Instructions and Procedures</b>	Review and concur with software Work Instructions or Procedures from subcontractor
<b>Planning</b>	Review and concur with subcontractor's SDP Annex and other software-related plans
<b>Cost/Schedule Reporting</b>	Review software subcontractor cost and schedule performance against baseline
<b>Requirements</b>	Responsible for requirement flowdown to software subcontractor
	Review and concur with software subcontractor SRS(s) and IRS(s),
<b>Architecture Design</b>	Review and concur software subcontractor's Software Architecture Description, SDD(s), IDD(s), and DBDD(s)
<b>Software Test</b>	Review and concur software subcontractor test approach, plans, and results
<b>Software Metrics</b>	Review all software metrics and ensure subcontractor takes corrective action when indicated by the metrics
<b>Problem Reports</b>	Address problems that affect functional performance
<b>Review Boards</b>	Participate in software subcontractor SCCB(s)



## AGI-2. Bibliography

Mandatory Implementation of Software Acquisition Process Improvement (SWAPI), Department of the Air Force, Memorandum for SMC-ALL, 22 March 2005.

Practical Software Measurement; Objective Information for Decision Makers, McGarry, J., Card, D., Jones, C., Layman, B., Clark, E., Dean, J., and Hall, F., Addison Wesley, October 2001.

Revitalizing the Software Aspects of Systems Engineering, Under Secretary of the Air Force, SAF/AQ Policy Memo 04A-003, 20 September 2004.

Software Acquisition Process Improvement Instruction, Space and Missile Systems Center Instruction 63-103, 1 December 2006.

Software Acquisition Instruction, Space and Missile Systems Center, Instruction 63-104, 28 June 2006.

Software Acquisition Management Plan (SWAMP) Preparation Guide, Eslinger, S., Gechman, M., Harralson, D., Holloway, L., and Sisti, F., The Aerospace Corporation Report TOR-2006(1455)-5743, 29 December 2006.

Software Development Standard for Space Systems, Adams, R., Eslinger, S., Hantos, P., Owens, K., Stephenson, L., Tagami, J., and Weiskopf, R., The Aerospace Corporation Report TOR-2004(3909)-3537, Rev. B, 11 March 2005. This standard was also republished as SMC Standard SMC-S-012, 13 June 2008.

Software Engineering—Software Measurement Process, International Organization for Standardization/International Electrotechnical Commission, ISO/IEC 15939, August 2007.

Software Measurement Standard for Space Systems, Abelson, L., Eslinger, S., Gechman, M., Korzec, K., LeDoux, C., Lieu, M., The Aerospace Corporation Report TOR-2009(8506)-6, 5 May 2011.

Standard for Information Technology; Software Life Cycle Processes; Software Development Acquirer—Supplier Agreement, EIA/IEEE Interim Standard, J-STD-016-1995, September 1995.

Technical Reviews and Audits for Systems, Equipments, and Computer Software, Space and Missile Systems Center Standard, SMC-S-21, 15 September 2009.

The Collaboration of Software and System Engineering, Gechman, M., The Aerospace Corporation Report TOR-2010(1475)-5, 27 June 2011.





### AGI-3. Software-Related Definitions

Note: Definition of terms contained in TOR-3537B are not repeated in this table.

Glossary	Definition
Algorithm Configuration Control Board	The ACCB is an interdisciplinary team of scientific and engineering personnel responsible for the approval and disposition of algorithm acceptance, verification, development and testing transitions.
Algorithm Software	Operational Algorithm Code – Algorithm code that has been verified to meet all functional and performance requirements for data quality, timeliness, and execution within the architecture.
Algorithm Team	Interdisciplinary team of scientific and engineering personnel assigned to the verification, development, and testing of a specific set of algorithms. Responsibilities include technical resolution of data quality or timeliness requirements issues.
Baselines	Software baselines describe a particular version of software (e.g., increment, build, or release) and consists of a set of internally consistent requirements, design, code, build files, and user documentation. A requirements baseline includes an SRS under CCB control.
Delivery	The process of providing a software product that is ready for acceptance by a higher-tier organization for the purpose of fulfilling a contractual requirement.
Element	A configuration item within a segment, consisting of integrated hardware and software. For the IDPS and C3S, an example of an element is a relocatable terminal; for the Space Segment, an example is a sensor payload.
Element (or Factory) Qualification Test	A set of formal criteria, such as a procedure, whose execution satisfies a set of requirements agreed to by an authorizing agency. EAT or FAT is performed at the contractor's software development facility.
Heritage	A previous baseline or baselines that current software may be based on.
Increment	A defined pass through the program lifecycle, including a sequential set of software lifecycle activities; may include multiple planned software builds.
Incremental Model	The incremental lifecycle model is a multi-build model. Software requirements analysis and architectural design, plus initial detailed design, code, integration, and test are completed in the first build. Additional capabilities are added in subsequent builds through detail design, code, integration, and test activities. This model supports delivery of an interim capability prior to the final software delivery.
Integration and Test (I&T)	Combines tested entities into the next higher entity (e.g., lower level SUs into higher level SUs), then tests the interactions between entities to verify the entities work correctly with each other, in accordance with test plans. Also verifies processes (e.g., tasks) synchronize correctly with processes in other components.
Legacy	The extent to which software may impact future programs or other software by nature of functionality or problems that may be introduced.
Modified Code	Code that has been previously constructed, and is being reused with modifications. See Reuse software.
Module	A text file containing source lines of code (SLOC). In C++ this is generally a single class.
Qualification	Testing that is performed in an environment functionally equivalent to the target environment and is intended to verify and validate all software requirements. Software Item Qualification Testing (SIQT) is done prior to acceptance testing. SIQT is a precursor to system/spacecraft-level test.
Release	The distribution of a new product or new function and fixes for an existing product. There are three types: 1) <u>Document Release</u> – after approval of a document by the CCB, the Data Center releases the document and posts it as a replacement of previous versions; 2) <u>COTS or Re-use Release</u> – an update to a product from the vendor. If, after review of the update, the updated product is accepted for further development, it is released to development; 3) <u>Software Release</u> – distribution of new versions of software to integration that include approved and tested changes.
Reuse	Reuse is any product which has been previously constructed and is utilized partially or completely, in the current development.
Reuse Software	Software is designated as "reuse software" if it has been previously constructed and is utilized, partially or completely, in the current development with less than 30 percent design breakage. If there is 30 percent or more design breakage, the software is considered "new."
Satellite	The spacecraft bus plus payload; synonymous with space segment.

Glossary	Definition
Segment	The collection of all Software Items and associated Hardware Items for each segment.
Software Engineering Notebook (SWEN)	A repository of program-unique information that provides an organized method of communicating technical information and providing a repository for historical data.
Sub-build	Development of a subset of the requirements allocated to a build. A sub-build goes through integration and test. Sub-builds follow the build process through integration and test, but not SIQT or beyond.
Tailoring	A process by which company software standards are mapped to the program's common software development process to ensure that the company requirements are being met by the program's common process or to waive specific company process requirements in favor of those needed for the program's common process.
Thread	An end-to-end functional capability traced through the system and verified by creating an input and observing the intended output.
Turnover	The process of providing a software product from one team member to another in accordance with pre-established quality criteria.
Work Instructions or Procedures	Documentation containing detailed directions for the day-to-day implementation of the software process. It is also referred to as a procedure. (see Table 8.3 for an example list.)
Work Package	The smallest element of work with allocated budget and schedule against which progress is reported and tracked.



### AGI-4. Software Acronyms

Acronym	Definition
AGI	Additional Guidebook Information
API	Application Programming Interface
APO	Acquisition Program Office
BAR	Build Architecture Review
BOE	Basis of Estimate
BTR	Build Turnover Review (or PTR or TER)
CAP	Corrective Action Plan
CAIV	Cost As An Independent Variable
CARD	Cost Analysis Requirements Document
CASE	Computer Aided Software Engineering
CBA-IPI	CMM <sup>SM</sup> -Based Assessment – Internal Process Improvement
CCB	Configuration Control Board
CCR	Critical Computer Resource
CDD	Capabilities Development Document
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CIP	Contract Implementation Plan
CM	Configuration Management
CMMI <sup>SM</sup>	Capability Maturity Model – Integrated
CMP	Configuration Management Plan
COM	Computer Operation Manual
CONOPS	Concept of Operations (see OCD)
COTS	Commercial Off-The-Shelf
CPI	Cost Performance Index
CPM	Computer Programming Manual
CPU	Central Processor Unit
C/R	COTS/Reuse (software)
CRF	Change Request Form
CSCI	Computer Software Configuration Item (see SI)
CSWE	Chief Software Engineer
CUT	Coding and Unit Testing
CWBS	Contract Work Breakdown Structure
DBDD	Data Base Design Description
DID	Data Item Description
DM	Data Management
DMS	Data Management System
DOORS	Dynamic Object-Oriented Requirements System (tool)
DR/CR	Discrepancy Report/Change Request
EAT	Element Acceptance Test (same as FAT)
ECR	Engineering Change Request
EDIN	Electronic Data Interchange Network
EMD	Engineering and Manufacturing Development
ERB	Engineering Review Board
ESLOC	Equivalent Source Line of Code
EVMS	Earned Value Management System
FAT	Factory Acceptance Test (see EAT)
FCA	Functional Configuration Audit
FMECA	Failure Modes, Effects, and Criticality Analysis
FOC	Full Operation Capability
FQT	Formal Qualification Test

Acronym	Definition
FSM	Firmware Support Manual
FSW	Flight Software
GFE	Government Furnished Equipment
GFS	Government Furnished Software
GOTS	Government Off-The-Shelf
GPS	Global Positioning Satellite/System
GSE	Ground Support Equipment
GUI	Graphical User Interface
HI	Hardware Item
HIQT	Hardware Item Qualification Test
HWCI	Hardware Configuration Item
I&T	Integration and Test
ICD	Initial Capabilities Document (see IFCD)
IDD	Interface Design Description
IDR	Interim Design Review
IEEE	Institute of Electrical and Electronics Engineers
IFCD	Interface Control Document
ILS	Integrated Logistics Support
IMF	Integrated Management Framework
IMP	Integrated Master Plan
IMS	Integrated Master Schedule
IA	Information Assurance
IORD	Integrated Operational Requirements Document
IPO	Input/Process/Output (flowchart)
IPPD	Integrated Process and Product Development
IPT	Integrated Product Team
IRD	Interface Requirements Document
IRR	Integration Readiness Review
IRS	Interface Requirements Specification
ISO	International Standards Organization
ITAR	International Traffic in Arms Regulation
JTR	Joint Technical Review
KDP	Key Decision Point
KSLOC	Thousand SLOC
LCC	Lifecycle Cost
MTP	Master Test Plan (formerly STEP)
MC	Mission Critical (software)
MMC	Mission Management Center
MOSA	Modular Open System Architecture
MSDL	Master Software Development Library
NDI	Non-Developmental Item
NSA	National Security Agency
OCD	Operational Concept Description
OOA/OOD	Object-Oriented Analysis/Design
PA	Product Assurance
PCA	Physical Configuration Audit
PDR	Preliminary Design Review
PDRR	Program Definition and Risk Reduction
PIP	Process Improvement Program
PM	Program Manager
PTR	Post Test Review (or BTR or TER)
PUI	Program Unique Identifier
QMP	Quantitative Management Plan

Acronym	Definition
QTRR	Qualification Test Readiness Review
RE	Responsible Engineer
RFP	Request For Proposal
RHP	Risk Handling Plan
RMP	Risk Management Plan
RTVM	Requirements Test Verification Matrix
S/W (SW)	Software
SAD	Software Architecture Description
SAR	SW Requirements and Architecture Review
SA/SD	System Analysis/System Design
SAT	Segment Acceptance Test
SC	Spacecraft
SCCB	Software Configuration Control Board
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SCOM	Software Center Operations Manual
SCR	Software Change Request (or Report)
SDCE	Software Development Capability Evaluation
SDD	Software Design Description
SDE	Software Development Environment
SDF	Software Development Folder (or File)
SDL	Software Development Library
SDP	Software Development Plan
SDR	System Design Review or Software Discrepancy Report
SDRL	Subcontractor Data Requirements List
SE&I	Systems Engineering and Integration
SEE	Software Engineering Environment
SEI	Software Engineering Institute
SEIT	System Engineering Integration and Test
SEMP	System Engineering Management Plan
SEN	Software Engineering Notebook
SEPG	Software Engineering Process Group
SETA	Systems Engineering and Technical Assistance
SFR	System Functional Review (see SDR)
SI	Software Item (see CSCI)
SIOM	Software Input/Output Manual
SIP	Software Installation Plan
SIQT	Software Item Qualification Test
SLATE	System-Level Automation Tool for Engineers
SLOC	Source Lines of Code
SMBP	Software Master Build Plan
SMP	Subcontract Management Plan or Software Maintenance Plan
SCMT	Subcontractor Management Team

Acronym	Definition
SOO	Statement Of Objectives
SOW	Statement Of Work
SPAR	Software Process Assets Repository
SPCR	Software Problem Change Report
SPE	Software Process Engineer
SPI	Schedule Performance Index
SPR	Software Peer Review
SPS	Software Product Specification
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
SQPP	Software Quality Program Plan
SQT	System Qualification Test
SRR	System Requirements Review
SRS	Software Requirements Specification
SRTM	Software Requirements Traceability Matrix
SS	Space Segment or Support Software
SSDD	System/Subsystem Design Description
SSP	Standard Software Process
SSPM	Software Standards and Practices Manual
SSR	SW Specification Review (see SAR)
SSS	System/Subsystem Specification
STD	Software Test Description
STE	Software Test Environment
ST&E	System Test and Evaluation
STEP	System Test and Evaluation Plan (see MTP)
STP	Software Test Plan
STR	Software Test Report (or Results)
STrP	Software Transition Plan
SU	Software Unit
SUM	Software Users Manual
SVD	Software Version Description (see VDD)
SWCCB	Software Change Control Board
SWED	Software Entity Database
TBX	To Be Reviewed, Determined, Supplied
TER	Test Exit Review (or BTR or PTR)
TIM	Technical Interchange Meeting
TPM	Technical Performance Measurement
TRD	Technical Requirements Document
TRM	Test Requirements Matrix
TRR	Test Readiness Review
TT&C	Telemetry, Tracking, and Control (or Command)
UI&T (UIT)	Unit Integration and Testing
UML	Unified Modeling Language
VCRM	Verification Cross Reference Matrix
VDD	Version Description Document (or SVD)
WBS	Work Breakdown Structure
WI	Work Instruction
XMPL	Example fictitious program (see Part 2, subsection 1.4)



## AGI-5. Subject Index to the SDP Guidebook

Subject	SDP Section, Subsection, Paragraph, Subparagraph
Access For Acquirer Review	4.2.8
Activities—Software Lifecycle	3.2.1
Activity Network	6.2
Analysis and Reporting	5.20.7
Analysis of User Input	5.3.1
Architectural Design—Software Item	5.6.2
Architectural Design—System/Segment	5.4.2
Architecture Overview	1.2.1, 1.2.2
As Built SI Design and Related Information—Preparation	5.13.4
Associate Developers—Coordination	5.24
Availability	4.2.5.4
Base Measures	5.20.6
Build Architecture Review	4.1.1
Build Functionality Matrix	5.1.1.3, 5.6
Build Planning/Requirements/Updating/Delivery	5.1.1.3, 5.5
Build Turnover Review	4.1.1
Categories	1.2.3
Change Control Board	5.17.1
Change Implementation	5.25.3
Change Request (or Report)	5.17.1
Chief Software Engineer	7.2.1.1, AGI-1 and AGI-2
Classes	1.2.3
Code and Unit Test	5.7
Code—Source	4.2.10, 5.7
Commercial Off-The-Shelf and Reuse Software	1.2.3.3, 4.2.4
Computer Aided Software Engineering (CASE) Tools	5.2.1
Computer Hardware Resource Utilization	4.2.6
Computer Operation Manual	5.12.3.2
Computer Programming Manual	5.13.8.1
Concept of Operations (see Operational Concepts Document)	5.3.2
Configuration Audits and Delivery	5.14.4, 5.14.5
Configuration Control Board	5.14
Configuration Identification/Control/Status Accounting	5.14.1, 5.14.2, 5.14.3
Configuration Management—Software	5.14, AGI-9
Configuration Management Plan (SCMP)	5.1, 5.14
Constraints—System	3.1
Constraints—Contractual and Non-Contractual	3.7, 3.8, 3.9
Contract Data Requirements List (CDRL)	3.6
Control Boards	5.17.2
Corrective Action System	5.17.2
Corrective Action/Corrective Action Plan	5.17
COTS/Reuse (Software)	1.2.3.3, 4.2.4
Critical Computer Resources	5.20.4
Critical Design Review	4.1.1, 5.6.3
Critical Requirements	4.2.5, 4.2.5.5
Data Management	4.2.9
Database—Standards	5.1.1
Deficiency Reporting	5.17
Definitions	Table 8.2
Dependability	4.2.5.4

Subject	SDP Section, Subsection, Paragraph, Subparagraph
Derived Metrics	5.20.6
Design Decisions—Software Item-Wide	4.2.7, 5.6.1
Design Decisions—System/Segment	5.4.1
Design Description (SDD)	4.2.10, 5.6
Design—Detailed Software Items	5.6.3
Design Process—System/Segment	5.4
Detailed Design—Software Items	5.6.3
Development Environment—Establishing	5.2
Development Facilities	7.2.2
Development Files (or Folders) (SDF)	5.2.4
Development Libraries	5.2.3
Development Methods	4.2.1
Development Planning	5.1.1
Development Process	4.1
Development Strategy—Requirements and Constraints	3.4, 3.4.1
Disaster Recovery	4.2.9.1
Document Peer Reviews	5.15, 5.15.2.1
Documentation Production and Constraints	3.3
Documents—Government and Non-Government	2.1, 2.2
DOORS—Dynamic Object-Oriented Requirements System	4.2.3
Earned Value Management System	5.1.1.4
Electronic Data Interchange Network	Part 1 (Section 3), 5.2.3.1, 5.5
Element (or Factory) Acceptance Test	3.7.2, 3.7.3, 3.7.4
Engineering Environment	5.2.1
Engineering Review Board	5.17.2
Equivalent Source Lines of Code (ESLOC)	1.2.3.3
Estimating Resources	5.1.1.2
Evaluations—Quality Assurance	5.16.1
Executable Software—Preparation	5.13.1
Facilities	5.2.1, 7.2.2
Firmware Support Manual	5.13.8.2
Functional Configuration Audit	3.7.4, 5.11
Government Furnished Equipment	7.2.3
Government Rights	4.2.9.2
Hardware Resource Utilization	4.2.6
Headcount Oversight	5.1.14
Implementation and Unit Testing	5.7
Improvement of Project Processes	5.25
Independence in Segment/System Qualification Testing	5.11.1
Independence in Software Item Qualification Testing	5.9.1
Indicators—Management Metrics/Measurement	5.20
Inspections—Formal	5.15.1.1
Installation at User Sites	5.12.4
Installation Planning/Installation Plan	5.1.4
Integrated Master Plan (IMP)	5.1.1
Integrated Master Schedule (IMS)	5.1.1; 5.1.14
Integrated Product Team (IPT)	Part 1 (Section 7), 5.7 through 5.11, 7.1, AGI-4 and AGI-5
Integration and Test Readiness Review	4.1.1
Integration and Testing—Preparation/Performing SI/Hi	5.10.1, 5.10.2
Integration and Testing—Software/Hardware Items	5.10
Integration and Testing—Analysis and Recording SI/Hi	5.10.4
Integration and Testing—Revision and Retesting SI/Hi	5.10.3



Subject	SDP Section, Subsection, Paragraph, Subparagraph
Integration—IT&V Approach/Objectives/Process	3.4.2, 3.4.3, 3.4.4
Interface Control Document (ICD)	5.3, 5.5
Interface Design Description (IDD)	5.6
Interface Requirements Specification (IRS)	5.5
Interface Specifications	5.3
Inter-group Coordination	5.27
International Traffic in Arms Regulation (ITAR)	4.2.9.3
Iterative Processes	4.1.3
IV&V Agents Interfacing	5.23
Joint Technical and Management Reviews	5.18, 5.18.1, 5.18.2
Key Technical Decisions	4.2.7
Level of Software Products	4.2.2
Libraries—Software Development (SDL)	5.2.3
Lifecycle Activities	3.2.1
Maintainability	4.2.5.4
Maintenance Manuals—Preparation	5.13.8, 5.26.2
Maintenance Plan/Manuals	5.13.8, 5.26.3
Maintenance Sites—Version Description and Preparation	5.13.3
Maintenance—Transition to	5.13
Management Indicators/Metrics/Measurement	5.20
Management Indicators—Analysis/Reporting/Thresholds	5.20.7
Management Indicators—Candidate Set of Metrics	5.20.5
Management Indicators—Continuous Improvement	5.20.3
Management Indicators—Management Approach	5.20.2
Management Indicators—Principal Objectives	5.20.1
Management Issues and Indicators	5.20.4
Management Plans	4.2.10.3
Management Reviews	5.18.2
Master Software Development Library (MSDL)	5.2.3
Measurement and Oversight	5.1.1.4, 5.20.4
Methods—Software Development	4.2.1
Metrics (see Management Indicators)	5.20
Mission Assurance	4.2.5.5
Mission Critical Software	1.2.3.1
Mission Critical Software Development Process	4.1.1
Models for the Development Process	4.1
Modular Open System Architecture	5.6.2
Non-Deliverable Software	5.2.5
Non-Document Work Products	4.2.10.2
Object-Oriented Analysis and Design	4.2.1
Operational Concept Description (OCD)	5.3.1, 5.3.2
Operations—Transition to	5.12
Operations and Maintenance	5.26
Organization of the Project	1.1; 7.1
Oversight	5.1.1.4
Packaging, Storage, Handling and Delivery	5.14.5
Peer Reviews/Prepare/Conduct/Analyze	5.15.1.1, 5.15.1.2, 5.15.1.3
Peer Reviews and Product Evaluations/Peer Review Plan	5.15
Personnel Resources	7.2.1
Physical Configuration Audit	3.7.4, 5.11
Planning Activities	5.1.1, 5.1.1.1
Planning—Installation/Transition	5.1.4, 5.1.5
Plans—Relationship Between	1.4

Subject	SDP Section, Subsection, Paragraph, Subparagraph
Plans—Following and Updating	5.1.6
Post Test Review	4.1.1
Preliminary Design Review	5.6.2
Privacy Protection	4.2.5.3, 5.21
Problem (or Software) Change Report (or Request)	5.17.1
Process Audits	5.25.2
Process—Development	4.1
Process Engineer/Lead	5.25.6, AGI-3
Process Improvement Process	5.25
Process Overview/Oversight	3.3, 5.1.1.4
Process—Mission Critical Software Development	4.1.1
Process—Support Software Development	4.1.2
Product Evaluations	5.15.2
Product Quality Assurance	5.16
Product Levels	4.2.2
Program Unique Identifier	5.1.1.3, 5.1.4.1, 5.9
Project Organization	7.1
Project Oversight/Overview	3.3, 5.1.1.4
Project Planning and Oversight	5.1
Project Resources	7.2
Proprietary Rights	4.2.9.2
Qualification Test—Analysis and Recording of Results	5.9.7
Qualification Testing—Software	5.9
Qualification Testing on the Target Computer System	5.9.2
Qualification Testing—Readiness Testing/Readiness Review	5.9.4
Qualification Testing—Performing	5.9.5
Qualification Testing—Preparation	5.9.3
Qualification Testing—Revision and Retesting	5.9.6
Quality Assurance—Software	5.16, AGI-10
Quality Assurance Evaluations and Records	5.16.1, 5.16.2
Quality Assurance—Independence	5.16.3
Quality Assurance—Non-Compliance Issues	5.16.4
Quality Control Plans	4.2.10.3
Quality Program Plan (SQPP)	5.16, 5.22
Quantitative Management Plan	5.20.7
Recording Rationale for Key Technical Decisions	4.2.7
Red Flags—Management Indicators	5.20.8
Reliability	4.2.5.4
Re-Planning	5.1.1.2
Request for Proposal (RFP)	5.3
Requirements Analysis—System/Segment	5.3
Requirements Analysis—Software	5.5
Requirements and Constraints	3.2
Requirements and Traceability Management (RTM tool)	4.2.3
Requirements Specification (SRS)	4.2.10, 5.5
Requirements—System/Segment	5.3.1, 5.3.3
Requirements Traceability Verification Matrix	5.5
Requisite Pro (tool)	4.2.3
Resource Estimating	5.1.1.2
Resource Utilization—Hardware	4.2.6
Resources—Acquirer-Furnished/Other Required	7.2.3, 7.2.4
Resources—Developer Facilities Overview	7.2.2
Resources—Personnel	7.2.1



Subject	SDP Section, Subsection, Paragraph, Subparagraph
Resources—Requirements and Constraints	3.5
Reusable Software Products	4.2.4
Reviews—Formal	4.1.1
Reviews—Joint Technical and Management	5.18.1, 5.18.2
Risk Handling Plan/Risk Mitigation Plan	4.2.4
Risk Management/Plan/Board	5.19
Roles and Responsibilities	3.3, 4.1.1, 4.1.2, Appendix A, 5.5 through 5.11, 7.2.2
Safety—Software	4.2.5.1
Schedule of Activities	6.1
Schedule—Requirements and Constraints	3.5
SDP Component Parts/Organization/Tailoring	Part 1, 1.3.1, 1.3.2
SDP Organization	1.3.2
SDP Overview	1.3
SDP Relationship to Other Plans	1.4
SDP Updates	1.3.3
Security and Privacy	4.2.5.2, 5.21
Segment Acceptance Testing	3.7.2, 3.7.3, 3.7.4, 5.11
Segment and System Qualification Testing	5.11
Segment/System Qualification Test—Analysis and Recording	5.11.7
Segment/System Qualification Testing—Dry Run	5.11.4
Segment/System Qualification Testing on the Target Computer	5.11.2
Segment/System Qualification Testing—Performing	5.11.5
Segment/System Qualification Testing—Preparation	5.11.3
Segment/System Qualification Testing—Revision and Retesting	5.11.6
Simulation and Modeling	5.4, 5.5, 5.6, 5.6.3, 5.7.2, 5.9, 5.9.3, 5.10.1, 5.11.3
Skill Levels	7.2.1.3
Small Software Developments	Part 1 (Section 11)
Software Architecture Description (SAD)	5.6
Software Center Operations Manual (SCOM)	5.12.3.3
Software Change Request (SCR)	5.17.1
Software Classes and Categories	1.2.3
Software Design/Software Design Description (SDD)	5.6
Software Development Files (or Folder)	5.2.4
Software Development Library	5.2.3
Software Development Plan (SDP)	Part 1, 1.3, 1.4
Software Discrepancy Report (SDR)	5.17.1
Software Engineering Environment (SEE)	5.2.1
Software Engineering Process Group (SEPG)	5.25.1, 7.1
Software Engineering Process Group—Infrastructure/Training	5.25.4, 5.25.5
Software Engineer—Responsibilities	AGI-7
Software Entity Database	5.1.1
Software Input/Output Manual (SIOM)	5.12.3.3
Software Item Lead—Responsibilities	AGI-6
Software Item Qualification Test (SIQT)/Test Planning	5.1.1, 5.1.2
Software Items Overview	1.1, 3.2.2
Software Process Assets Repository (SPAR)	5.2.3
Software Product Specification (SPS)	3.3, 5.12.1, 5.13.4
Software Requirements and Constraints	3.2
Software Requirements Specification (SRS)—Updating	4.2.10, 5.13.6, 5.5
Software Safety	4.2.5.1, 5.26

Subject	SDP Section, Subsection, Paragraph, Subparagraph
Software Standards and Practices Manuals	4.2.2.1
Software Test Description (STD)	4.2.10, 5.7 through 5.11
Software Test Environment	5.2.2
Software Test Plan (STP)	4.2.10, 5.7 through 5.11
Software Test Report (STR)	4.2.10, 5.7 through 5.11
Software Transition Plan (STrP)	3.3, 5.12, 5.13.9
Software Version Description—(see VDD)	4.2.10, 5.12.2
Source Code	4.2.10
Source Files—Preparation	5.13.2
Staffing/Staff Loading	5.1.1.2, 7.2.1.2
Standards and Practices for Software Products/Manuals	4.2.2, 4.2.2.1
Statement of Objectives (SOO)	5.3
Subcontract Management/Management Team	5.22, AGI-11
Subcontractor—Compliance With the SDP	5.22
Support Software—Development Process	1.2.3.2, 4.1.2
Sustainment—Sustainment Organization	5.26, 5.26.4
Sustainment Planning	5.26.2
System Architecture Overview	1.2.1
System Engineering Integration and Test (SEIT)	5.3, 5.4
System Functional Review	4.1.2, 5.4
System Overview	1.2
System Qualification Testing	5.11
System Requirements and Constraints	5.3
System Requirements Review	4.1.2, 5.3
System Requirements—Updating	5.13.7
System Test and Evaluation Plan	5.10
System Test Planning	5.1.3
System/Segment Design	5.4, 5.4.2
System/Segment Requirements Analysis	5.3, 5.3.3
System/Subsystem Design Description—Updating	5.13.5
System-Level Automation Tool for Engineers (SLATE)	4.2.3
Tailoring of the SDP	Part 1 (Section 10)
Team Responsibilities	3.2.2
Technical Interchange Meetings (TIMs)	4.1.2, 5.18, 5.6.2
Technical Performance Measurements (TPM)	4.2.6
Technical Requirements Document (TRD)	3.5, 5.3
Technical Reviews	5.18.1
Test Documentation	4.2.10
Test Engineer Responsibilities	AGI-8
Test-Like-You-Fly	5.2.2
Test Planning—Software Items and Environment	5.1.2, 5.2.2
Test Planning—System Level	5.1.3
Test Readiness Review (TRR)	4.1, 5.9.4, 5.9.5, 5.10.2, 5.11.5
Testing—IT&V Objectives/Approach/Approach	3.7.2, 3.7.3, 3.7.4
Thresholds—Management Indicators/Metrics	5.20.8
Tools	5.2.1
Traceability	4.2.3
Tracking and Oversight	5.1.1.4
Training Plans	7.2.5
Training Process	5.2.5.5
Transition Planning	5.1.5
Transition to Maintenance	5.13
Transition to Operations	5.12



Subject	SDP Section, Subsection, Paragraph, Subparagraph
Transition to Operations—Preparing Executable Software	5.12.1
Transition to the Designated Maintenance Site	5.13.9
Unified Modeling Language	4.2.1
Unit Integration and Testing	5.8
Unit Integration and Testing—Preparation and Performing	5.8.1, 5.8.2
Unit Integration and Testing—Analysis and Recording Results	5.8.4
Unit Integration and Testing—Revision and Retesting	5.8.3
Unit Test Results—Analysis and Recording	5.7.5
Unit Testing—Preparation/Performing/Revision/Retesting	5.7.2, 5.7.3, 5.7.4
Updating Software Requirements	5.13.6
User Input Analysis	5.3.1
User Manuals and Guides	4.2.10, 5.12.3
User Sites Installation	5.12.4
Verification Reviews	5.15.1
Verification—IT&V Approach/Objectives/Process	3.7.2, 3.7.3, 3.7.4
Version Description Document (same as SVD)	4.2.10
Version Descriptions for maintenance sites—Preparation	5.13.3
Version Descriptions for User Sites	5.12.2
Waiver Processing	5.1.1
Work Breakdown Structure	5.1, 6, 7.1, 7.2
Work Instructions	4.2.10.2, Table 8.3
Work Products	4.2.10, 5.5 through 5.9

