Software Development Standard for Mission Critical Systems

March 17, 2014

Richard. J. Adams¹, Suellen Eslinger², Karen L. Owens¹, and Joanne M. Tagami¹
¹Software Acquisition and Process Department, Software Engineering Subdivision
²Software Engineering Subdivision, Computers and Software Division

and

Michael A. Zambrana
Directorate of Systems Engineering
Space and Missile Systems Center, USAF

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

Contract No. FA8802-14-C-0001

Authorized by: Space Systems Group

Distribution Statement A: Approved for public release; distribution unlimited.
**Report Documentation Page**

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

<table>
<thead>
<tr>
<th>1. REPORT DATE</th>
<th>2. REPORT TYPE</th>
<th>3. DATES COVERED</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 MAR 2014</td>
<td>Final</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. TITLE AND SUBTITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Development Standard for Mission Critical Systems</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6. AUTHOR(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richard. J. Adams, Suellen Eslinger, Karen L. Owens, and Joanne M. Tagami and Michael A. Zambrana</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Aerospace Corporation 2310 E. El Segundo Blvd. El Segundo, CA 90245-4609</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8. PERFORMING ORGANIZATION REPORT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR-RS-2015-00012</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space and Missile Systems Center Air Force Space Command 483 N. Aviation Blvd. El Segundo, CA 90245-2808</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10. SPONSOR/MONITOR’S ACRONYM(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>11. SPONSOR/MONITOR’S REPORT NUMBER(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12. DISTRIBUTION/AVAILABILITY STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved for public release, distribution unlimited</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>13. SUPPLEMENTARY NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>14. ABSTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update of software development standard.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15. SUBJECT TERMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software, space systems</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16. SECURITY CLASSIFICATION OF:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. REPORT unclassified</td>
</tr>
<tr>
<td>b. ABSTRACT unclassified</td>
</tr>
<tr>
<td>c. THIS PAGE unclassified</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>17. LIMITATION OF ABSTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>UU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>18. NUMBER OF PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19a. NAME OF RESPONSIBLE PERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18
This report was submitted by The Aerospace Corporation, El Segundo, CA 90245-4691, under Contract No. FA8802-14-C-0001 with the Space and Missile Systems Center, 483 N. Aviation Blvd., El Segundo, CA 90245. It was reviewed and approved for The Aerospace Corporation by Andrew B. Dawdy, principal director, Engineering Directorate, and Asya Campbell, principal director, Software Engineering Subdivision. David E. Davis was the project officer.

This report has been reviewed by the Public Affairs Office (PAS) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication. Publication of this report does not constitute Air Force approval of the report’s findings or conclusions. It is published only for the exchange and stimulation of ideas.

David E. Davis
SMC/EN

© The Aerospace Corporation, 2014.

All trademarks, service marks, and trade names are the property of their respective owners.
Software Development Standard for Mission Critical Systems

Richard J. Adams, Suellen Eslinger, Karen L. Owens, Joanne M. Tagami, and Michael A. Zambrana

The Aerospace Corporation
2310 E. El Segundo Blvd.
El Segundo, CA 90245-4691

TR-RS-2015-00012

Approved for public release; distribution unlimited.

This TOR is a standard for the software development process that is applied on contracts for mission critical systems. This report provides a full lifecycle software development process standard. This version includes an updated template, similar to a Data Item Description (DID), for the Software Development Plan (SDP) consistent with the updated standard. It also includes templates for the Software Architecture Description (SAD), Software Master Build Plan (SMBP), Software Measurement Plan (SMP), Software Measurement Report (SMR), and Process Improvement Plan (PIP). See the Change Log for a list of major changes from previous versions of the Software Development Standard.
## Change Log

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Sections</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rev A</strong></td>
<td>11 March 2005</td>
<td>5.1.1</td>
<td>Reference to Appendix H instead of SDP DID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.2</td>
<td>Replaced SDP DID with “Use Appendix H.”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Appendix H</td>
<td>Added Appendix H SDP Template.</td>
</tr>
<tr>
<td><strong>Rev B</strong></td>
<td>11 March 2005</td>
<td>None</td>
<td>Distribution became public unlimited. No content change was made. Date remains the same.</td>
</tr>
<tr>
<td>(Editor provided November 2005)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Foreword and Introduction</td>
<td>Deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Many Incorporated tailoring and lessons learned from recent acquisitions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>All All requirements were assigned unique identifiers within their subsections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>All All multipart requirements were split and assigned unique identifiers within their subsections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Most Deleted all extra shall statements that merely required other shall statements to be met. (No double counting.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Updates to acquisition- and contract-related terms.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 Deletions, additions, and updates to referenced documents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.1 Additions and updates to software terms for clarification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.2.3 Updates and additions to software bidirectional traceability. All traceability requirements were gathered in this subsection.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.2.4 Updates and additions to reusable software products.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.2.5 Updates and additions to assurance of critical requirements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.2.8 Updates and additions to access for acquirer review.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.2.9 Added contractual requirements for software.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.1 and 5.2 Updates to software test planning and integration and test environments.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.3 Updated requirements for system requirements analysis.</td>
</tr>
<tr>
<td>Issue</td>
<td>Date</td>
<td>Sections</td>
<td>Changes</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>5.4</td>
<td></td>
<td></td>
<td>Updated requirements for system architectural design.</td>
</tr>
<tr>
<td>5.5</td>
<td></td>
<td></td>
<td>Updated requirements for software requirements analysis.</td>
</tr>
<tr>
<td>5.6</td>
<td></td>
<td></td>
<td>Major update to software design to add software architecture and rewrite software design.</td>
</tr>
<tr>
<td>5.7 – 5.11</td>
<td></td>
<td>Major updates to improve the robustness of software testing at several levels from unit testing to system qualification testing.</td>
<td></td>
</tr>
<tr>
<td>5.12</td>
<td></td>
<td></td>
<td>Updated requirements for transition to operations.</td>
</tr>
<tr>
<td>5.13</td>
<td></td>
<td></td>
<td>Updated requirements for transition to maintenance.</td>
</tr>
<tr>
<td>5.14</td>
<td></td>
<td></td>
<td>Added requirements for configuration management baselines and clarified requirements for other configuration management aspects.</td>
</tr>
<tr>
<td>5.15</td>
<td></td>
<td></td>
<td>Updated and added requirements for peer reviews.</td>
</tr>
<tr>
<td>5.16</td>
<td></td>
<td></td>
<td>Updated and added requirements for quality assurance.</td>
</tr>
<tr>
<td>5.17</td>
<td></td>
<td></td>
<td>Updated requirements for corrective action.</td>
</tr>
<tr>
<td>5.18</td>
<td></td>
<td></td>
<td>Updated requirements for joint technical and management reviews.</td>
</tr>
<tr>
<td>5.20</td>
<td></td>
<td></td>
<td>Replaced requirements for software measurement.</td>
</tr>
<tr>
<td>5.22</td>
<td></td>
<td></td>
<td>Replaced requirements for subcontractors with requirements for software team members.</td>
</tr>
<tr>
<td>5.24</td>
<td></td>
<td></td>
<td>Updated requirements for coordination with associate developers.</td>
</tr>
<tr>
<td>5.25</td>
<td></td>
<td></td>
<td>Added requirements for process improvement and Process Improvement Plan.</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>Updated information for current acquisition practices. Included and updated information from old Appendix G. Added references to new templates in Appendix H.</td>
</tr>
<tr>
<td>Appendix A</td>
<td></td>
<td></td>
<td>Updated acronym list.</td>
</tr>
<tr>
<td>Appendix B</td>
<td></td>
<td></td>
<td>Made mandatory and updated evaluation criteria for reusable software.</td>
</tr>
<tr>
<td>Appendix C</td>
<td></td>
<td></td>
<td>Made mandatory and added discrepancy and change report requirements.</td>
</tr>
<tr>
<td>Appendix D</td>
<td></td>
<td></td>
<td>Reformatted for easier understanding and added more product review criteria.</td>
</tr>
<tr>
<td>Appendix E</td>
<td></td>
<td></td>
<td>Made mandatory and added build reviews and objectives.</td>
</tr>
<tr>
<td>Appendix F</td>
<td></td>
<td></td>
<td>Replaced guidance for software indicators with mandatory test log requirements.</td>
</tr>
<tr>
<td>Appendix G</td>
<td></td>
<td></td>
<td>Deleted all content.</td>
</tr>
<tr>
<td>Appendix H</td>
<td></td>
<td></td>
<td>Made mandatory and updated Software Development Plan template (H.1) to match updates to standard and provide project information. Added templates for Software Architecture Description, Software Master Build Plan, Software Measurement Plan, Software Measurement Report, and Process Improvement Plan.</td>
</tr>
<tr>
<td>Issue</td>
<td>Date</td>
<td>Sections</td>
<td>Changes</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.2.5.6 #6a</td>
<td>Removed “validating” other software.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.1</td>
<td>Changed definitions of discrepancy, software development file, and added definition of demonstrate, stakeholder, and turnover. Clarified definitions of a few other terms, e.g., software development file, target computer system, test procedures.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.2.3</td>
<td>Clarified bidirectional traceability.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.9.3</td>
<td>Updated and clarified advance notice for software qualification testing and related events.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.15.1</td>
<td>Simplified peer review data references.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Appendix E</td>
<td>Added advance notification to acquirer for technical reviews.</td>
</tr>
</tbody>
</table>
Contents

Change Log .......................................................................................................................................... v

1. Scope ................................................................................................................................................. 1
   1.1 Purpose ...................................................................................................................................... 1
   1.2 Application .............................................................................................................................. 1
      1.2.1 Organizations and Agreements ...................................................................................... 1
      1.2.2 Contract-Specific Application ...................................................................................... 1
      1.2.3 Tailoring .......................................................................................................................... 1
      1.2.4 Compliance ...................................................................................................................... 1
      1.2.5 Interpretation of Selected Terms .................................................................................... 2
   1.3 Method and Tool Independence .............................................................................................. 4

2. Referenced Documents .................................................................................................................... 5

3. Definitions ......................................................................................................................................... 7
   3.1 Terms ......................................................................................................................................... 7

4. General Requirements ..................................................................................................................... 21
   4.1 Software Development Process ............................................................................................ 21
   4.2 General Requirements for Software Development ............................................................ 22
      4.2.1 Software Development Methods .................................................................................. 22
      4.2.2 Standards for Software Products ............................................................................... 22
      4.2.3 Traceability ...................................................................................................................... 22
      4.2.4 Reusable Software Products ....................................................................................... 26
      4.2.5 Assurance of Critical Requirements ........................................................................... 26
      4.2.6 Computer Hardware Resource Utilization .................................................................. 27
      4.2.7 Recording Rationale ....................................................................................................... 27
      4.2.8 Access for Acquirer Review .......................................................................................... 27
      4.2.9 Contractual Requirements for Software ....................................................................... 28

5. Detailed Requirements .................................................................................................................... 29
   5.1 Project Planning and Oversight ............................................................................................. 29
      5.1.1 Software Development Planning .................................................................................. 29
      5.1.2 Software Integration and Qualification Test Planning ................................................ 30
      5.1.3 System Qualification Test Planning ............................................................................. 31
      5.1.4 Planning for Software Transition to Operations .......................................................... 31
      5.1.5 Planning for Software Transition to Maintenance ....................................................... 31
      5.1.6 Following and Updating Plans ...................................................................................... 32
   5.2 Establishing a Software Development Environment ............................................................. 32
      5.2.1 Software Engineering Environment ............................................................................ 32
      5.2.2 Software Integration and Qualification Test Environment ......................................... 33
      5.2.3 Software Development Library ..................................................................................... 33
      5.2.4 Software Development Files .......................................................................................... 33
      5.2.5 Nondeliverable Software ............................................................................................... 33
   5.3 System Requirements Analysis ............................................................................................... 33
      5.3.1 Analysis of User Input ................................................................................................... 34
      5.3.2 Operational Concept ...................................................................................................... 34
      5.3.3 System Requirements Definition .................................................................................. 34
   5.4 System Architecture and Design ............................................................................................ 34
      5.4.1 System-wide Architectural Design Decisions .............................................................. 35
      5.4.2 System Architectural Design ......................................................................................... 35
   5.5 Software Requirements Analysis ............................................................................................. 35
   5.6 Software Architecture and Design .......................................................................................... 36
5.13.4 Preparing the ‘As Built’ Software Architecture, Design, and Related Information .......................... 63
5.13.5 Updating the System/Subsystem Design Description ................................................................. 63
5.13.6 Updating the Software Requirements ....................................................................................... 64
5.13.7 Updating the System Requirements ....................................................................................... 64
5.13.8 Preparing Maintenance Manuals ............................................................................................. 64
5.13.9 Transition to the Designated Maintenance Site(s) .................................................................. 65
5.14 Software Configuration Management ........................................................................................... 65
5.14.1 Configuration Identification ...................................................................................................... 65
5.14.2 Configuration Control ............................................................................................................... 66
5.14.3 Configuration Status Accounting ............................................................................................ 67
5.14.4 Configuration Audits ............................................................................................................... 67
5.14.5 Packaging, Storage, Handling, and Delivery ......................................................................... 67
5.14.6 Baselines ................................................................................................................................ 68
5.15 Software Peer Reviews and Product Evaluations ........................................................................... 68
5.15.1 Software Peer Reviews ............................................................................................................ 69
5.15.2 Product Evaluations ................................................................................................................ 70
5.16 Software Quality Assurance ....................................................................................................... 71
5.16.1 Software Quality Assurance Evaluations ............................................................................... 71
5.16.2 Software Quality Assurance Records .................................................................................... 72
5.16.3 Independence in Software Quality Assurance ....................................................................... 72
5.16.4 Software Quality Assurance Noncompliance Issues ............................................................ 72
5.17 Corrective Action .......................................................................................................................... 73
5.17.1 Discrepancy and Change Reports ........................................................................................... 73
5.17.2 Corrective Action System ........................................................................................................ 73
5.18 Joint Technical and Management Reviews .................................................................................... 74
5.18.1 Joint Technical Reviews ........................................................................................................... 74
5.18.2 Joint Management Reviews .................................................................................................... 74
5.19 Software Risk Management ........................................................................................................ 75
5.20 Software Measurement ................................................................................................................ 75
5.20.1 Software Measurement Planning ........................................................................................... 75
5.20.2 Software Measurement Reporting .......................................................................................... 76
5.20.3 Software Measurement Working Group (SMWG) ................................................................. 76
5.21 Security and Privacy ..................................................................................................................... 76
5.22 Software Team Member Management ........................................................................................... 76
5.23 Interface with Software IV&V Agents .......................................................................................... 76
5.24 Coordination with Associate Developers .................................................................................... 76
5.25 Improvement of Project Processes .............................................................................................. 76
6. Notes ........................................................................................................................................... 79
6.1 Intended Use ................................................................................................................................ 79
6.2 Data Item Descriptions (DIDs) ..................................................................................................... 79
6.3 Deliverable Versus Nondeliverable Software Products .................................................................. 80
6.3.1 Philosophy of the Standard ...................................................................................................... 80
6.3.2 Contracting for Deliverables ..................................................................................................... 80
6.3.3 Scheduling Deliverables .......................................................................................................... 80
6.3.4 Format of Deliverables .............................................................................................................. 80
6.5 Tailoring Guidance ........................................................................................................................ 80
6.6 Related Standardization Documents ............................................................................................ 81
Appendix A. List of Acronyms and Abbreviations .............................................................................. A-1
A.1 Scope ......................................................................................................................................... A-1
A.1.1 Acronyms and Abbreviations .................................................................................................. A-1
H.1 Software Development Plan (SDP) Template ................................................................. H.1-1
H.2 Software Architecture Description (SAD) Template ...................................................... H.2-1
H.3 Software Master Build Plan (SMBP) Template .............................................................. H.3-1
H.4 Software Measurement Plan (SMP) Template ............................................................ H.4-1
H.5 Software Measurement Report (SMR) Template ........................................................ H.5-1
H.6 Process Improvement Plan (PIP) Template .................................................................... H.6-1

Appendix I. Document Improvement Proposal ................................................................. I-1
I.1 Recommended Changes ...................................................................................................... I-1

Tables

Table C.2-1 Categories of Software Products ......................................................................... C-4
Table C.2-2 Categories of Activities ....................................................................................... C-4
Table C.2-3 Categories of Severity ......................................................................................... C-5
Table C.2-4 Categories of Causes .......................................................................................... C-5
Table C.2-5 Discrepancy and Change Report Information ...................................................... C-6
Table D.3-1 Products and Associated Evaluation Criteria .................................................... D-4
Table E.5-1 Software Management Products ....................................................................... E-13
1. Scope

1.1 Purpose
The purpose of this standard is to establish uniform requirements for software development activities for mission critical systems.
Note 1: Terms that appear in italics are defined in Section 1.2 or Section 3.1 Terms.
Note 2: This standard is based on (DOD MIL-STD-498).

1.2 Application
This standard applies to the development of mission critical systems that contain software (such as hardware-software systems), software-only systems, and stand-alone software products. The application of this standard is intended as in the following paragraphs.

1.2.1 Organizations and Agreements
This standard can be applied to contractors or acquirer in-house agencies performing software development. Within this standard, the term “acquirer” is used for the organization requiring the technical effort (see Section 3.1); the term “developer” is used for the organization(s) performing the technical effort (see Section 3.1); the term “contract” is used for the agreement between these parties (see Section 3.1); the term “Statement of Work” (SOW) is used for the list of tasks to be performed by the developer; and the term “Contract Data Requirements List” (CDRL) is used for the list of deliverable products.

1.2.2 Contract-Specific Application
This standard is invoked by citing it on a contract as a compliance document. It applies to each product and to each category of software covered by the contract, regardless of storage medium. The acquirer is expected to specify the categories of software to which the standard applies and to tailor the standard appropriately for each category of software. While this standard is written in terms of software items, it applies to software within the covered categories of software that is not designated as a software item, with the term “software item” interpreted appropriately. This standard applies to software installed in firmware devices.

This standard applies to the prime contractor and all software team members. This standard applies to the categories of software defined in Section 1.2.5.

1.2.3 Tailoring
This standard can be tailored for each category of software to which it is applied. General tailoring guidance can be found in Section 6.5.

1.2.4 Compliance
1.2.4.1 Compliance Definition
Compliance with this standard as tailored for a project and recorded in the contract is defined as:
1. Performing the activities that resulted from tailoring this standard for a specific project, and
2. Recording applicable information resulting from the performance of the activities.
An activity is complete when all actions constituting the activity, as specified in the contract, have been accomplished and all applicable information has been recorded.

1.2.4.2 Compliance Terminology
For this standard, the words “shall,” “should,” and “may” are reserved words used to designate different levels of compliance. The term “shall” is used in statements of mandatory requirements. The term “should” is used to indicate a goal that the developer should attempt to adhere to, if at all

---

1This subsection is based on information from Section 1 of (EIA/IEEE J-016).
possible. The term “may” is used to indicate an optional method of satisfying a requirement. Use of the simple present and future tenses is limited to descriptive information only. In particular, the terms “will” and “must” are not used in this standard for any level of compliance.

### 1.2.4.3 Order of Precedence

In the event of conflict between the requirements of this standard and other mandatory standards, the acquirer is responsible for resolving the conflicts.

### 1.2.5 Interpretation of Selected Terms

Terms that appear in italics are defined in Section 1.2 Application or Section 3.1 Terms. The following terms have a special interpretation as used in this standard.

#### 1.2.5.1 Interpretation of ‘System’ and ‘Subsystem’

The following interpretations of the words “system” and “subsystem” apply:

1. The term “system,” as used in this standard, means:
   a. A software-hardware system (for example, a radar system) for which this standard covers only the software portion, or
   b. A software system (for example, a mission planning system, or software running on the acquirer’s computers) for which this standard governs overall development.

2. The term “subsystem” is used to mean any system component between the entire system and the individual software items and hardware items. Depending on context, these components could be called subsystems, segments, elements, prime items, critical items, complex items, or other acquirer-specific terminology. For example, a system could consist of segments; a segment could consist of subsystems or elements; and a subsystem or element could consist of software items and hardware items.

3. If a system consists of subsystems, all requirements in this standard concerning systems apply to the subsystems as well. If a contract is based on alternatives to systems and subsystems, such as segments or complex items, the requirements in this standard concerning the system and its specification apply to these alternatives and their specifications.

#### 1.2.5.2 Interpretation of ‘Participate’ in System Development

The term “participate” in sections regarding system or subsystem activities is to be interpreted as follows:

1. If the software covered by this standard is part of a hardware-software system for which this standard covers only the software portion, the term “participate” depends upon what organization is in charge of the activity:
   a. If the activity is performed by the software organization, then “participate” is to be interpreted as “be responsible for.”
   b. If the activity is performed by another organization such as an integration and test organization, then “participate” is to be interpreted as “take part in.”

2. If the software (possibly with its computers) is considered to constitute a system, the term “participate” is to be interpreted as “be responsible for.”

3. If the software in a hardware-software system is modified, but the hardware is not, the term “participate” is to be interpreted as “be responsible for.”

#### 1.2.5.3 Interpretation of ‘Develop,’ ‘Define,’ Etc.

Throughout this standard, requirements to “develop,” “define,” “establish,” or “identify” information are interpreted to include new development, modification, reuse, reengineering, maintenance, or any other activity or combination of activities resulting in products. Within this standard, requirements to “develop,” “define,” “establish,” or “identify” always include recording the information even if that is not explicitly stated. These terms also include maintaining the previous and current information, software products, facilities, infrastructure, software development environments, capability, etc.
throughout the system development lifecycle. Maintaining the development environments includes such activities as maintaining configuration management over all components of the development environment; keeping the configurations current; keeping the versions of reusable software, including commercial off-the-shelf (COTS) software, current so that supplier support is maintained; and keeping the plans and procedures current.

1.2.5.4 Interpretation of ‘Record’ or ‘Document’ (the Verbs)
Throughout this standard, requirements to “record” or “document” information are interpreted to mean “set down in a manner that can be retrieved and viewed.” The result can take many forms, including, but not limited to, hard copy or electronic documents and data recorded in computer-aided engineering (CAE) and project management tools. This interpretation also applies to implicit requirements to record or document as described in Section 1.2.5.3.

1.2.5.5 Interpretation of ‘Applicable’
Within this standard, requirements to provide “applicable” information or to perform “applicable” activities are interpreted as meaning to “provide the information or perform the activities that are required for the project in accordance with the methods, tools, and procedures explained in the Software Development Plan.” The term “applicable” suggests that not all development projects will require the information or the activity. The acquirer and the developer jointly agree upon whether providing information or performing activities are applicable.

1.2.5.6 Interpretation of ‘Categories of Software’
The standard applies to the following categories of software including the software portion of firmware:

1. software onboard space vehicles (e.g., spacecraft bus, communications, payload);
2. software onboard launch and upper-stage vehicles;
3. ground operations software (e.g., mission planning; mission data processing; mission data distribution; mission data storage; mission support; telemetry, tracking, and commanding; infrastructure and services);
4. user equipment software (e.g., software embedded in ground terminals, handheld receivers, receivers onboard wheeled vehicles, spacecraft, aircraft, ships, and weapons);
5. software that allows configuration management of initial and updated data, including constants, command sequences and procedures, and computer instructions, stored in onboard, ground, and user equipment databases or other types of data stores, and software that supports the verification and validation of this data; and
6. other software (e.g., applications, security, safety, training, modeling, simulation, analysis, database support, automatic test equipment, test facility and environment, and maintenance) used in any of the following:
   a. satisfying or verifying requirements, or
   b. performing or supporting operations.

1.2.5.7 Interpretation of ‘User’
Within this standard, the term “users” includes operators of the system in addition to the end users of the data produced by the system.

1.2.5.8 Interpretation of ‘User Site’
Within this standard, the term “user site” is interpreted to include individual ground operations sites, individual space vehicles, individual launch vehicles, individual mobile ground systems, and individual types of user equipment. This term is used primarily in Section 5.12.
1.2.5.9 **Interpretation of ‘Prime Contractor’**
Within this standard, the term “prime contractor” is interpreted to mean the organization with which the acquirer has the prime contract for the project. The prime contractor can also be the developer.

1.2.5.10 **Interpretation of ‘Subcontractor’**
Within this standard, the term “subcontractor” is interpreted to mean any software team member tasked by the prime contractor or another software team member to perform part of the required software-related effort. This definition is broader than the usual legal definition of subcontractor.

1.3 **Method and Tool Independence**
This standard is independent of any particular software development methods or tools. No requirement in this standard mandates that a particular method or tool be used.
2. Referenced Documents


SMC-S-012. Space and Missile Systems Center Standard, SMC-S-012, Software Development Standard. This is the same as this document.


3. Definitions

3.1 Terms

The terms defined in this section appear in *italics* in the rest of the standard when they are not used as adjectives.

**Acquirer.** An organization that procures products for itself or another organization.

**Agile development lifecycle model.** A group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, and a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle. Source: Adapted from (Wikipedia). See also evolutionary, incremental, iterative, spiral, and waterfall development lifecycle models.

**Anomaly.** See discrepancy.

**Applicable.** See Section 1.2.5.5.

**Approval.** Written notification by an authorized representative of the acquirer that a developer’s plans, requirements, designs, or other aspects of the project appear to be sound and can be used as the basis for further work. Such approval does not shift responsibility from the developer to meet contractual requirements.

**Architecture.** The fundamental organization of the system or software embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. Source: Adapted from (IEEE 1471)

**Associate developer.** An organization that is neither prime contractor, software team member, nor subcontractor to the developer, but that has a development role on the same or related system or project.

**Automatically generated code.** Source code that is automatically generated by tools such as model-based software development tools, graphical user interface (GUI) builders, form builders, and computer-aided software engineering (CASE) tools.

**Availability.** Prior to the start of a mission, the probability that the system will be mission capable when a mission is needed. Availability accounts for system faults and the time it takes to restore the system to a mission capable state following failure. Source: (TAI SPD)

**Bandwidth.** A measure of available or consumed data communication resources. Examples of types of data communications resources include network, data bus, ground-ground link, ground-space link. Source: (Wikipedia)

**Baseline.** A product or a set of work products that has been formally reviewed and agreed on at a particular point in the item’s lifecycle, which thereafter serves at the basis for further development, and which can be changed only through change control procedures. Source: Adapted from (SEI 2010) Note: A software baseline can be a single product or a set of consistent products for a particular build that can include, e.g., requirements, architecture, design, source code files and the associated executable code, build files, test plans, test cases, test procedures, test results, and user documentation. Source: Adapted from (SEI 2010)

**Behavioral design.** The design of how an overall system or a software item will behave, from a user’s point of view, in meeting its requirements, ignoring the internal implementation of the system or the
software item. This design contrasts with architectural design, which identifies the internal components of the system or the software item, and with the detailed design of those components.

Bidirectional traceability. A two-way relationship between two products; for example, the relationship between a requirement and the design of a given software component, and the relationship between the design of the software component and the requirements it satisfies. Source: Adapted from (IEEE 610.12)

Build. A version of software that meets a specified subset of the requirements that the completed software will meet.

Note: The relationship of the terms “build” and “version” is up to the developer; for example, it can take several versions to reach a build, a build might be released in several parallel versions (such as to different sites), or the terms can be used as synonyms. Other terms that developers often use as synonyms include: block, cycle, drop, increment, iteration, and spiral. This standard uses the term “build” for all of these.

Categories of software. See Section 1.2.5.6.

Change request. A request for a change to products, documents, or processes. Change requests might result from a discrepancy, a new requirement, a changed requirement, or an improvement suggestion.

Characterization testing. Testing software or a system to explore and determine its behavioral, performance, and other characteristics. For example, the capabilities and capacity of an unfamiliar legacy or COTS product need to be analyzed to determine its suitability for use or whether it still behaves the same way after changes.

Checkout. The process after software has been installed at a user site or maintenance site to exercise the software to demonstrate that the software behaves as expected or required. Checkout procedures might use subsets of the regression test suite.

Child requirement. A requirement in a child specification that can be traced upward to one or more requirements in a specification in the specification tree immediately above that child specification. A child requirement may also be a derived requirement. See parent requirement.

Child specification. A specification in the specification tree that is immediately below another specification in the specification tree. See parent specification. Child specifications contain child requirements.

Commercial off-the-shelf (COTS) software. See reusable software.

Note: COTS software is sometimes called “commercial item” software. “Commercial item” is defined in the FAR 2.101. Source: (FAR)

Component. A constituent part. Source: (Merriam-Webster)

Computer database. See database.

Computer hardware. Devices capable of accepting and storing computer and sensor data, executing a systematic sequence of operations on computer data, or producing control outputs. Such devices can perform substantial interpretation, computation, communication, control, or other logical functions.

Computer program. A combination of computer instructions and data definitions that enable computer hardware to perform computational or control functions. The computer instructions and data definitions in a computer program can exist as source code, object code, or executable code. Source: (IEEE 610.12)

Computer software. See software.

Computer software configuration item (CSCI). See software item.
Computer software unit (CSU). See software unit.

Contract. The agreement between the acquirer and the developer. See Section 1.2.1.

Contract Data Requirements List (CDRL). See Section 1.2.1.

Contractual requirement. A mandatory statement in this standard or another portion of the contract.

Cyber-security. A property of cyberspace that is an ability to resist intentional and unintentional threats, respond, and recover. Source: (Russia-U.S.)

Cyberspace. An electronic medium through which information is created, transmitted, received, stored, processed, and deleted. Source: (Russia-U.S.)

Database. A collection of related data stored in one or more computerized files in a manner that can be accessed by users or computer programs via a database management system.

Database management system. An integrated set of computer programs that provides the capabilities needed to establish, modify, make available, and maintain the integrity of a database.

Data item description (DID). The format and content preparation instructions for a data product generated by the specific and discrete task requirements as delineated in the contract. Examples include DIDs for the Software Test Plan (STP), Software Test Report (STR), and Software User Manual (SUM) and templates for the Software Development Plan (SDP), Software Architecture Description (SAD), and Software Measurement Plan (SMP). See Section 6.2 for a list of DIDs and templates applicable to this standard.

Define. See Section 1.2.5.3.

Deliverable product. A product that is required by the contract to be delivered to the acquirer and other acquirer-designated recipients.

Demonstrate. To prove or make clear by evidence. Adapted from (Merriam Webster)

Note: Demonstrating always includes recording the information even if that is not explicitly stated. This definition of “demonstrate” is not to be confused with the verification method of “demonstration.”

Dependability. The ability: a) to deliver service that can justifiably be trusted, and b) to avoid service failures that are more frequent and more severe than is acceptable. Dependability is an integrating concept that encompasses the following attributes:

1. readiness for correct service;
2. continuity of correct service;
3. absence of catastrophic consequences on the user(s) and the environment;
4. absence of improper system alterations; and
5. ability to undergo modifications and repairs. Source: (Avizienis 2004)

Derived requirement. A system or software requirement that results from architecture or design decisions or from the user’s operational concepts and that is not directly traceable to one or more higher level requirements. A derived requirement may be either a functional or nonfunctional requirement.

Design. Those characteristics of a system or software item that are selected by the developer in response to the requirements. Some characteristics match the requirements; other characteristics are elaborations of requirements, such as definitions of all error messages in response to a requirement to display error messages; and others are implementation related, such as decisions about what software units and logic to use to satisfy the requirements.

Develop. See Section 1.2.5.3.
**Developer.** An organization that develops software products (“develops” includes new development, modification, integration, reuse, reengineering, maintenance, or any other activity that results in products). The term “developer” encompasses all software team members. See software team member.

**Developer-internal software item integration testing.** The last stage of software unit integration and testing where all of the components of the software item are integrated and tested. If the software item is developed in multiple builds, then the developer-internal software item integration testing occurs on a build-by-build basis.

**Developer-internal system integration testing.** The last stage of hardware-software integration and testing where all of the components of the system are integrated and tested. If the system is developed in multiple builds, then the developer-internal system integration testing occurs on a build-by-build basis.

**Discrepancy.** Any condition that deviates from expectations based on requirements specifications, architecture documents, design documents, user documents, plans, procedures, reports, standards, policy, etc., or from a user’s or other stakeholder’s sound engineering judgment or experiences. Discrepancies can be found during, but not limited to, the review, test, analysis, compilation, or use of products or applicable documentation. The term discrepancy is used throughout this standard where others might use the terms: anomaly, defect, error, fault, failure, incident, flaw, problem, gripe, glitch, or bug. Source: Adapted from the term “anomaly” in (IEEE 1044).

**Discrepancy and change report (DCR).** Documentation of change requests, discrepancies, and test incidents. Discrepancy and change reports are written for any potential change, discrepancy, or test incident, even if the final resolution is that no discrepancy exists, or if the change request is rejected. Discrepancy and change reports are sometimes called problem reports, change requests, trouble reports, test incident reports, issue reports, and other terms.

**Document (as a verb).** See Section 1.2.5.4.

**Document (as a noun) or documentation.** A collection of data, regardless of the medium on which it is recorded, that generally has permanence and can be read by humans or machines.

**Duration testing.** See endurance testing and stability testing. Contrast with stress testing.

**End-to-end functional capability.** A series of one or more software or system functions that:

1. satisfy a set of related requirements (or allocated portions of requirements);
2. exercise the software or system from inputs to the software or system through the software and hardware, interacting with databases, sensors, communications, and other software or system components, as applicable; and
3. result in outputs from the software to the user, another software or system component, or another system.

Note: The term “end-to-end functional capability,” as used in this standard, applies only to software unit integration and testing (see Section 5.8.2), software item qualification testing (see Section 5.9.3), and software-hardware item integration and testing (see Section 5.10.2), and it does not apply to system qualification testing.

The scope of an end-to-end functional capability differs depending upon the entity under test. For software unit integration and testing, an end-to-end functional capability is defined by inputs to one or more software units, processing and data flows that cross unit boundaries, and outputs from one or more software units, that together satisfy the portion of software or software interface requirements allocated to the units under test. For software item qualification testing, an end-to-end functional capability is defined by inputs to the software item, processing by the software item, data flows through the software item, and outputs from the software item that together satisfy one or more
**Endurance testing.** Testing over prolonged execution time or elapsed time to determine whether longer-term execution shows functional, performance, or other problems. Note: This testing often checks for clock overflows, buffer overflows, memory leaks, or other discrepancies that do not necessarily occur in shorter tests. Also known as duration testing or stability testing. Source: Adapted from (Cohen 2005). Contrast with stress testing.

**Ensure.** To make sure, certain, or secure by performing action(s). Source: Adapted from (Merriam-Webster).

**Equivalence class.** An input set ("class") in which all elements cause the same ("equivalent") execution path, regardless of which element from the class is chosen.

**Establish.** See Section 1.2.5.3.

**Evaluation.** The process of determining whether an item or activity meets specified criteria.

**Evolutionary development lifecycle model.** A software development lifecycle model in which requirements analysis and definition are done in each build. Each build can contain architecture, design, implementation, integration, and testing in an overlapping, iterative manner, resulting in evolutionary requirements and incremental completion of the overall software product. Each build is not necessarily delivered to the acquirer. See also agile, incremental, iterative, spiral, and waterfall development lifecycle models. Source: Adapted from (IEEE 610.12)

**Expected results.** The desired or required results specified in a test case or test procedure or both. Types of expected results include, but are not limited to, output data, responses from the software, status, colors, displays, warnings, alarms, events, or error messages. Expected results should include the units, range, accuracy, and precision of the output data, as applicable.

**Extreme values.** Values (of test case inputs) that are maximum values, minimum values, or the value zero for the data type of the test case inputs, whether or not those values are within the expected range.

Note: Extreme values might be discontinuous.

**Firmware.** The combination of a hardware device with computer instructions or computer data or both that reside as read-only software on the hardware device.

**Functional requirement.** A requirement that defines a specific behavior or function of the system or software. Contrast with nonfunctional requirements that specify criteria that can be used to judge the operation of a system rather than specific behaviors. In general, functional requirements define what a system is supposed to do, while nonfunctional requirements define how a system is supposed to be.

**Hardware item.** An aggregation of hardware that satisfies an end use function and is designated for specification, interfacing, qualification testing, configuration management, or other purposes.

**High-fidelity simulator.** A simulation that provides static and dynamic behavior representative of the object being modeled with, as a minimum, nominal and off-nominal conditions, stressing conditions, and erroneous conditions that could occur during a mission. Some of these conditions include, but are not limited to: correct format, incorrect format, error injection, correct timing, incorrect timing, normal volume of data, extensive volume of data, more data than the system can handle, higher data needs.
rates than the system can handle, lost data blocks, extra data blocks, error data, changing data, fluctuating data, erroneous data, and any combination of the above.

**Human factors engineering.** The application of human behavior, abilities, limitations, and other characteristics to the design of *systems* for effective human use.

**Identify.** See Section 1.2.5.3.

**Incremental development lifecycle model.** A *software development lifecycle model* in which all software requirements analysis and definition occurs first, followed by a series of *builds* in which *architecture, design, implementation, integration,* and *testing* occur in an overlapping, iterative manner, resulting in incremental completion of the overall software *product*. Each *build* is not necessarily delivered to the *acquirer*. See also *agile, evolutionary, iterative, spiral,* and *waterfall development lifecycle models*. Source: Adapted from (IEEE 610.12)

**Independent verification and validation (IV&V).** Systematic *evaluation* of *products, activities,* or both by an agent that is not responsible for developing the *product* or performing the activity being evaluated. IV&V is not within the scope of this standard, but interfacing with IV&V agents is within the scope of this standard.

**Information assurance.** Measures that protect and defend information and information systems by ensuring their *availability, integrity, authentication, confidentiality,* and *nonrepudiation*. These measures include providing for restoration of information systems by incorporating protection, detection, and reaction capabilities. Information assurance ensures that the correct information is provided to the correct individuals at the correct time, in other words, that accurate information is shared only with those authorized to access it, and is available when it is needed. Source: (CNSSI 4009 2010)

**Integral process.** A *process* that supports the product-oriented development *processes* throughout the software lifecycle. Integral processes are performed concurrently with the product-oriented *processes* and are essential to their quality and completion. Examples of integral processes are the software peer review *process,* software measurement *process,* software quality assurance *process,* and software configuration management *process.*

**Interface.** In *software development,* a relationship among two or more entities (such as *software item* to *software item,* *software item* to *hardware item,* *software item* to user, *software unit* to *software unit*) in which the entities share, provide, or exchange data. An interface is not a *software item,* *software unit,* or other *system component;* it is a relationship among them.

**Issue.** A matter that is in dispute or is undecided. After investigation an issue could be resolved or could result in a *risk, problem,* or *discrepancy.*

**Iterative development lifecycle model.** A *software development lifecycle model,* such as *incremental,* *evolutionary,* or *spiral* with repeating and possibly overlapping development activities. Note: Iterative development also includes agile development. See also *agile, evolutionary, incremental, spiral,* and *waterfall development lifecycle models*. Source: Adapted from (IEEE 610.12)

**Joint review.** A *process* or meeting involving representatives of both the *acquirer* and the *developer,* during which project status, *products,* and project *issues* are examined and discussed.

**Key performance parameter (KPP).** Those minimum attributes or characteristics considered most essential for an effective mission capability. Source: (DAU Glossary)

**Maintain.** See Section 1.2.5.3.

**Maintainability.**
1. The probability that the system can be retained in or returned to a mission capable state within a specified period of time when a failure occurs. See software maintenance.

2. The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. See software maintenance.  
Source: (IEEE 610.12)

Maintenance (of software). See software maintenance.

Maintenance organization. The organization that is responsible for modifying and otherwise sustaining the software and other software products and documentation after transition from the development organization (e.g., a separate organization within a developer company or a different organization).

May. See Section 1.2.4.2.

Nominal conditions. Those conditions that are within the range anticipated in the requirements and design (e.g., operator workloads, processor loads, memory utilizations, input and output data rates, timing and sequencing of input data, interface error rates, power, temperature, and related parameters affecting the computing run-time platform and environment).

Nominal values. Those values within the expected ranges, types, ordering, consistency, and completeness anticipated in the computer hardware and software requirements and design.

Nondeliverable product. A product that is not required by the contract to be delivered to the acquirer or other designated recipient.

Nonfunctional requirement. A system or software requirement that specifies criteria that can be used to judge the operation of the system or software, rather than specific behaviors. Contrast with functional requirements that define specific behavior or functions. In general, nonfunctional requirements define how a system is supposed to be, while functional requirements define what a system is supposed to do. Nonfunctional requirements are often called “qualities” of a system. Other terms for nonfunctional requirements are “constraints,” “quality attributes,” “quality goals,” “quality of service requirements,” and “nonbehavioral requirements.” Performance requirements, including response times, throughput, and hard time deadlines, are considered nonfunctional requirements.

Nonfunctional requirements can be divided into two main categories:
1. execution requirements, such as security and usability, which are observable at run time; and
2. evolution requirements, such as testability, maintainability, extensibility, and scalability, which are embodied in the static structure of the software. Source: Adapted from (Wikipedia)

Off-nominal conditions. Conditions that are outside of the range of nominal conditions (see nominal conditions).

Off-nominal tests. Tests of off-nominal conditions and values, the purpose of which is to determine whether the software or system behaves in a manner that enables it to remain in normal operations or recovery to normal operations.

Off-nominal values. Values that are outside the range of nominal values (see nominal values). For example, if the acceptable range of a temperature value is between 0° C and 70° C, inclusive, then temperature values less than 0° C and greater than 70° C are off-nominal.

Open source software (OSS). Software for which the human-readable source code is available for use, study, reuse, modification, enhancement, and redistribution by the users of that software. In other words, OSS is software for which the source code is “open.” Source: (DOD-CIO 2009)

Operational hardware. Hardware in the operational system that is not part of the target computer system. Such hardware may have embedded software that is part of the hardware item but is not a separate software item. Changes to operational hardware include changes to its embedded software.
**Parent requirement.** A requirement in a parent specification that can be traced downward to one or more requirements in specifications in the specification tree immediately below that parent specification. See child requirement.

**Parent specification.** A specification in the specification tree that is immediately above one or more other specifications in the specification tree. See child specification. Parent specifications contain parent requirements.

**Participate.** See Section 1.2.5.2.

**Performance requirement.** See nonfunctional requirement.

**Prime contractor.** See Section 1.2.5.9.

**Privacy protection.** Absence of a weakness or failure that could lead to a breach of system privacy protection or of personally identifiable information. Source: (Avizienis 2004)

**Process.** A set of interrelated activities, which transform inputs into outputs, to achieve a given purpose, for example, the software development process. Source: (SEI 2010)

**Product.** Information created, modified, or incorporated by means of software or system development processes. Examples include plans, requirements, architecture, design, code, databases, test information, and manuals.

**Prototype.** A preliminary version of part of the hardware or software of a system that serves as a model for later stages or for the final, complete version of the system. Source: Adapted from (IEEE 610.12)

**Prototyping.** A hardware and software development technique in which a preliminary version of part of the hardware or software is developed early in the development process to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process. Source: Adapted from (IEEE 610.12)

**Qualification testing.** Testing performed to verify that a system or software item meets its specified requirements.

**Quality attribute.** See nonfunctional requirement.

**Record (as a verb).** See Section 1.2.5.4.

**Redline.** To mark up a document to show changes, either with a pen (usually a red one), or electronically using a track changes feature. The term “redline” is also used for the resulting changes.

**Reengineering.** The process of examining and altering an existing system to reconstitute it in a new form. It could include reverse engineering (analyzing a system and producing a representation at a higher level of abstraction, such as design from source code), restructuring (transforming a system from one representation to another at the same level of abstraction), redocumentation (analyzing a system and producing user or support documentation), forward engineering (using products derived from an existing system, together with new requirements, to produce a new system), retargeting (transforming a system to install it on a different target computer system), and translation (transforming source code from one language to another or from one version of a language to another).

**Regression testing.** Selective retesting of a system or component to determine whether modifications to the system or its environment (e.g., operating system, software, or hardware) have caused unintended effects and to determine whether the system or component still complies with its specified requirements. Source: Adapted from (IEEE 610.12)
Regression test suite. A set of test cases, test procedures, test drivers, test data, test databases, and instructions for preparing the test environment that is used to determine whether any changes adversely impact the functioning or performance of the software or system or adversely impact the previously verified requirements. See regression testing.

Reliability. The probability that the system will be able to complete the mission without failure, given that the system was available at the start of the mission.

Representative set. A set of values selected from an equivalence class that is representative of the distribution of values or conditions. The size of the set, i.e., the quantity of the values, within each equivalence class is dependent on the software item or system in question and the level of confidence needed for successful execution during nominal and off-nominal conditions.

Requirement.
1. A mandatory characteristic of a system or software item.
2. A mandatory statement in this standard or another portion of the contract.
Note: Software requirements also include software interface requirements.

Reusable software. Software developed for one use but having other uses, or developed specifically to be usable on multiple projects or in multiple roles on one project. Each use could include all or part of the software and could involve its modification. Examples of reusable software include, but are not limited to:
1. pre-existing developer software, including product-line software;
2. software in reuse libraries;
3. acquirer-furnished software;
4. open source software (OSS); and
5. commercial off-the-shelf (COTS) software.
Note: Software documentation products (for example, requirements, architectures, design, test cases) can also be reused, but they are not included in this definition of reusable software.

Risk. The potential for a negative future reality. Source: (DOD DAU 2001), Ch. 15

Safety. Absence of conditions that can cause a hazardous system state, i.e., one that could result in unintended death, injury, occupational illness, damage to or loss of property, environmental harm. Source: Adapted from (Avizienis 2004)

Security.
1. Security ensures that a system can deliver the correct information to the correct individuals at the correct time. This includes the concurrent existence of:
   a. availability for authorized actions only;
   b. confidentiality, i.e., the absence of unauthorized disclosure of information; and
   c. integrity, i.e., absence of unauthorized system alterations. Source: (Avizienis 2004)
2. A condition that results from the establishment and maintenance of protective measures that enable an enterprise to perform its mission or critical functions despite risks posed by threats to its use of information systems. Protective measures may involve a combination of deterrence, avoidance, prevention, detection, recovery, and correction that should form part of the enterprise’s risk management approach. Source: (CNSSI 4009 2010)
Note: For computer systems, security includes information assurance and cyber-security.

Shall. See Section 1.2.4.2.

Should. See Section 1.2.4.2.

Software. Computer programs, procedures, and data pertaining to the operation of a computer system. Data includes, for example, information in databases, rule bases, and configuration data. Procedures
Software development. A set of activities that results in software products. Software development includes new development, modification, reuse, reengineering, maintenance, and any other activities that result in software products.

Software development file (SDF). A repository for material pertinent to the development of a particular body of software. Contents typically include (either directly or by reference) the development products themselves as well as considerations, rationale, and constraints related to requirements analysis, architecture, design, and implementation; test information, including test cases and test results; discrepancy and change reports; technical reports; notes; and schedule and status information. Source: Adapted from (IEEE 610.12) Note: The body of software may be at different levels of integration (e.g., software unit, collection of related software units, software build, software item).

Software development library (SDL). A controlled collection of software, documentation, other intermediate and final software products, and associated tools and procedures used to facilitate the orderly development and subsequent maintenance of software.

Software development lifecycle model. A project management framework:
1. containing the activities involved in the development, operation, and maintenance of the software; and
2. spanning the life of the software from the definition of its requirements to the termination of its use.

See agile, evolutionary, incremental, iterative, spiral, and waterfall development lifecycle models. Source: Adapted from (IEEE 610.12) See (IEEE 1074) for more information.

Software development process. An organized set of activities performed to translate user needs into software products.

Software engineering. In general usage, a synonym for software development. As used in this standard, a subset of software development consisting of all activities except integration and qualification testing. The standard makes this distinction for the sole purpose of giving separate names to the software engineering environment and software integration and qualification test environment.

Software engineering environment. The facilities, hardware, software, firmware, procedures, and documentation needed to perform software engineering, including, but not limited to, tools to develop and analyze requirements, architect, design, develop, control, generate, verify, and load the software. Elements include, but are not limited to, computer-aided software engineering (CASE) tools, requirements management tools, architecture modeling tools, compilers, assemblers, linkers, loaders, code analyzers, path and coverage analyzers, operating systems, debuggers, simulators, emulators, configuration management tools, discrepancy and change reporting tools, measurement tools, unit test case generators and other unit test tools, documentation tools, and database management systems. Note: There may be multiple software engineering environments for various team members and sites.

Software integration and qualification test environment. The facilities, hardware, software, firmware, procedures, and documentation needed to perform integration, qualification and possibly other testing of software. Elements include, but are not limited to, hardware, test software, test drivers, automated test equipment, test beds, instrumentation, test case generators, automated test result checking tools, other test analysis tools, simulators, emulators, and databases and database values. Elements used in the software engineering environment can also be used in the software integration and qualification
test environment. Note: There may be multiple integration and qualification testing environments for different team members, levels of integration and testing, and sites.

**Software item.** An aggregation of *software* that satisfies an end use function and is designated for *specification*, *interfacing*, *qualification testing*, configuration management, and other purposes. Software items are selected based on tradeoffs among software function, size, host or *target computer systems*, *developer*, support strategies, plans for reuse, criticality, interface considerations, the need to be separately documented and controlled, and other factors. A software item is composed of one or more *software units*. A software item is sometimes called a computer software configuration item (CSCI). Source: Adapted from (EIA/IEEE J-016)

**Software item qualification test environment.** The facilities, hardware, *software*, *firmware*, procedures, and *documentation* needed to perform *qualification testing* of the *software*. This is usually a subset of the *software integration and qualification test environment*.

**Software maintenance.** The set of activities that takes place to ensure that *software* installed for operational use continues to perform as intended and fulfill its intended role in system operation. Software maintenance includes software corrections and improvements, user assistance, and related activities. Source: Adapted from (EIA/IEEE J-016)

**Software quality.** The ability of *software* to satisfy its specified *requirements* and expectations.

**Software support.** See *software maintenance*.

**Software-only system.** A *system* consisting solely of *software* and possibly the computer equipment on which the *software* operates.

**Software team member.** The *prime contractor* or any internal or external organization that *develops*, *tests*, integrates, or supports software-related work being performed for the *contract* and that has a formal or informal agreement with the *prime contractor* or any other team member. These organizations include, but are not limited to, intra-corporation organizations, in-house service providers, *developers*, fabrication and manufacturing organizations, laboratories, joint venture partners, teaming partners, subsidiaries, interdivisional transfer, and corporations that have a legal *contract* with the *prime contractor* or other team member. Examples of an agreement include a *contract*, work authorization, memorandum of agreement, or oral agreement. This standard includes all software team members under the term *developer*.

**Software transition.** The set of activities that enables responsibility for *software* to pass from one organization to another.

**Software unit.** An element in the *design* of a *software item*, for example, a major subdivision of a *software item*, a *component* of that subdivision, a class, object, module, function, routine, or *database*. Software units might occur at different levels of a hierarchy and might consist of other software units. Software units in the *design* might or might not have a one-to-one relationship with the code and data entities (e.g., routines, procedures, *databases*, data files) that implement them or with the computer files containing those entities. A software unit is sometimes called a computer software unit (CSU).

**Source code.** Computer instructions and data definitions expressed in a form suitable for input to an assembler, compiler, or other translator. Source: (IEEE 610.12)

**Specification.** A description of the verifiable technical *requirements* and design constraints for hardware and *computer software*, materials, and processes along with the *verification method* for determining whether each *requirement* is met. Source: (Shaw 2013)

**Specification tree.** The hierarchy of *system components* for which *requirements* are defined. This hierarchy of *system components* is usually depicted as a tree diagram.
Spiral development lifecycle model. A software development lifecycle model in which the selection of constituent activities, typically requirements analysis and definition, architecture, design, prototyping, coding, integration, and testing, are performed for each spiral based on the information that is needed to reduce risk. After each spiral, the new information and the remaining risks help the stakeholders select the activities for the next spiral. These spirals continue until the software is complete. See also agile, evolutionary, incremental, iterative, and waterfall development lifecycle models. Source: Adapted from (IEEE 610.12)

Stability testing. See endurance testing.

Stakeholder. A group or individual that is affected by or is in some way accountable for the outcome of a product. Adapted from (SEI 2010)

Statement of Work (SOW). See Section 1.2.1.

Stress testing. Testing using worst-case scenarios (e.g., extreme workloads, high frequency of inputs and events, large number of users, simulated failed hardware, missing or malfunctioning interfaces, tight timelines) to determine whether these aspects show functional, performance, or other problems. Contrast with duration testing, endurance testing, and stability testing.

Subcontractor. See Section 1.2.5.10.

Subsystem. See Section 1.2.5.1.

Supplier. As used in this standard, a software team member that provides reusable software products. This includes vendors of COTS products.

Support (of software). See software maintenance.

Supportability. The degree to which system design characteristics and planned logistics resources, including software, meet mission readiness and operational utilization requirements before the mission begins.

System. See Section 1.2.5.1.

Target computer system (TCS). The computer hardware, including all computer-related hardware such as processors, memory, storage, networks, and data interfaces, on which the operational software will reside and execute. The TCS used for software item qualification testing (SIQT) (and possibly for software-software and software-hardware integration testing) is an exact duplicate of the actual TCS that will be used in mission operations. The TCS contains the exact same model(s) of the processor(s) with the exact same instruction set(s) and timing as is used in mission operations. It should be from the same batch of processors.

Note 1: For space systems during testing, the TCS is called the “flight-like target computer system.” The only difference is that it is not the TCS that goes through all of the environmental tests.

Note 2: For some ground systems during testing, the TCS is called the “site-identical computer system” or the “site-identical target computer system.”

Note 3: For some ground systems, representative networks and a subset of target computer systems may be used for a case where the final deployed networks and configurations are not accessible to the test team. In this case, one or more procedures from the qualification test suite need to be re-executed at the user site for:

1) performance requirements,
2) concurrent access by users across the network, and
3) worst cases.

Technical performance measure (TPM). A measurement that indicates progress toward meeting critical system characteristics (technical parameters) that are specified in requirements or constrained by system design. Technical parameters that are tracked with TPMs have clearly identifiable and
measurable target and threshold values. Examples of software technical parameters, which can be tracked using TPMs, are computer system resource margins (e.g., central processing unit (CPU), input and output (I/O) volume or rates, memory margins) and response times.

Test. The terms “test” and “testing,” as used in this standard, refer to the activities of verifying that the implementation meets the design (unit and integration testing) and verifying that the requirements are satisfied (qualification testing). These terms are distinct from the verification method “Test,” which is only one of the four standard methods used for verification, i.e., Inspection (I), Analysis (A), Demonstration (D), and Test (T). See verification method.

Testability. The degree to which:

1. a requirement is stated in terms that permit establishment of test criteria and performance of tests to determine whether those criteria have been met, or
2. a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met. Source: Adapted from (IEEE 610.12)

Test activity suspension. A temporary interruption of testing activities from which recovery can be made without corrections to software, hardware, test procedures, test environment, or configuration. Criteria for suspension could include, e.g., test interruptions and test incidents.

Test activity termination. A halt to testing activities until a future date after necessary corrections to software, hardware, test procedure, test environment, or configuration have been made and checked out. Test activity termination usually requires a test dry run before resumption of test activities. Criteria for termination could include, e.g., 10 percent of the nominal cases fail.

Test case. A set of items for a test. Each test case consists of the verification method, requirements to be verified, inputs, prerequisite conditions, expected results, and criteria for evaluating results.

Test incident. Any event occurring during the execution of a software test that requires investigation. Source: Adapted from (IEEE 610.12)

Test interruption. Any interruption of testing. Examples of causes include: computer crash, incorrect configuration discovered, equipment or hardware problem or failure, simulator problem, power failure, biological break, water pipe break, exercise, evacuation (e.g., fire alarm, drill, tornado, hurricane, earthquake).

Test procedure. The collection of execution and analysis steps, including expected results, that are to be performed in order to obtain the test results needed to verify the requirements. Each test case has one or more associated test procedures.

Transition (of software). See software transition.

Turnover. The event when operations officially starts using the system or a new version of the system.

Usability. The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. Usability is a nonfunctional requirement. Source: Adapted from (ISO 9241-11)

User. See Section 1.2.5.7.

User site. See Section 1.2.5.8.

Validation. Confirmation that the product or service, as provided (or as it will be provided), will fulfill its intended use. Validation ensures that “you built the right thing.” (See verification.) Source: (SEI 2010)
Verification. Confirmation that work products and products properly reflect the requirements specified for them. Verification ensures that “you built it right.” (See validation.) Source: Adapted from (SEI 2010)

Verification method. One of four techniques used to fully or partially verify a requirement: Inspection (I), Analysis (A), Demonstration (D), and Test (T). These techniques are also called “qualification methods.” These four techniques are defined as follows:

1. **Inspection.** A method used to determine characteristics by inspecting engineering documentation produced during product development, including both hardware and software documentation, or by inspection of the product itself to verify conformance with specified requirements. Inspection generally is nondestructive and consists of visual inspections or simple measurements without the use of precision measurement equipment.

2. **Analysis.** A method used to verify requirements by determining qualitative and quantitative properties and performance by studying and examining engineering drawings, software and hardware flow diagrams, software and hardware specifications, and other hardware and software documentation (e.g., COTS vendor documentation), or by performing modeling, simulation, or calculations, or any combination, and analyzing the results. Similarity analysis is used in lieu of tests or demonstrations when it can be shown that an item is similar to or identical in design to another item that has been certified previously to equivalent or more stringent criteria.

3. **Demonstration.** A method used to verify requirements by exercising or operating the system or a part of the system in which instrumentation or special test equipment is not required beyond that inherently provided in the system being verified. In the demonstration method, sufficient data for requirements verification can be obtained by observing functional operation of the system or a part of the system. When this verification method generates data that are recorded by inherent instrumentation, inherent test equipment, or operational procedures, any analysis that must be performed using the data collected during the demonstration is an integral part of this method and should not be confused with the analysis method of verification described above.

4. **Test.** A method used to verify requirements by exercising or operating the system or a part of the system using instrumentation (hardware or software or both) or special test equipment that is not an integral part of the system being verified. The test method by its nature generates data, which are recorded by the instrumentation, test equipment, or procedures. Analysis or review is performed on the data derived from the testing. This analysis, as described here, is an integral part of this method and should not be confused with the analysis method of verification described above. Source: (Adams 2002)

**Waterfall development lifecycle model.** A software development lifecycle model in which the constituent activities, typically a concept phase, requirements phase, architecture phase, design phase, implementation phase, test phase, and installation and checkout phase, are performed in that order, possibly with overlap but with little or no iteration. See also agile, evolutionary, incremental, iterative, and spiral development lifecycle models. Source: Adapted from (IEEE 610.12)

**Work product.** Part or all of a product produced by means of the activities in this standard. A work product can be a component of a product but is not necessarily a deliverable product.
4. General Requirements

1. This standard shall apply to all software team members.
   Note: The prime contractor is considered a software team member (see Section 3).
2. This standard shall apply to all categories of software defined in Section 1.2.5.6 that are within the scope of the contract.
   Note: This standard applies to software included in the categories of software whether or not the software is identified as a software item.
3. This standard shall apply to software installed in firmware devices.
   Note 1: This standard does not apply to the hardware element of firmware.
   Note 2: For more detailed requirements for Application-Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs), see (Sather 2010) and (Dixon 2006).

4.1 Software Development Process

The framework used to organize the major software activities is called the software development lifecycle model.

1. The developer shall select software development lifecycle model(s) appropriate to the software being developed.
   Note: A software project can have more than one software development lifecycle model in use for the different categories of software being developed. (See Section 3.1 for definitions of agile, evolutionary, incremental, iterative, spiral, and waterfall lifecycle models.)
2. The developer shall record the selected software development lifecycle model(s) in the Software Development Plan (SDP) template in Appendix H.1 Software Development Plan Template.
   Note: See Section 6.2 for the SDP template identifier.
3. The developer shall provide in the SDP a description of each software development lifecycle model that will be used on the project.
4. The developer shall define in the SDP the following for each software development lifecycle model to be used:
   a. the portion of the system development lifecycle where it will be used,
   b. under which circumstances it will be used, and
   c. for which software items it will be used.
5. The developer shall establish a software development process consistent with contract requirements.
6. The software development process shall include the following major activities and integral processes:
   a. Project Planning and Oversight (Section 5.1)
   b. Establishing a Software Development Environment (Section 5.2)
   c. System Requirements Analysis (Section 5.3)
   d. System Architectural Design (Section 5.4)
   e. Software Requirements Analysis (Section 5.5)
   f. Software Architecture and Design (Section 5.6)
   g. Software Implementation and Unit Testing (Section 5.7)
   h. Unit Integration and Testing (Section 5.8)
   i. Software Item Qualification Testing (Section 5.9)
   j. Software-Hardware Item Integration and Testing (Section 5.10)
   k. System Qualification Testing (Section 5.11)
   l. Preparing for Software Transition to Operations (Section 5.12)
   m. Preparing for Software Transition to Maintenance (Section 5.13)
Integral Processes:

1. Software Configuration Management (Section 5.14)
2. Software Peer Reviews and Product Evaluations (Section 5.15)
3. Software Quality Assurance (Section 5.16)
4. Corrective Action (Section 5.17)
5. Joint Technical and Management Reviews (Section 5.18)
6. Software Risk Management (Section 5.19)
7. Software Measurement (Section 5.20)
8. Security and Privacy (Section 5.21)
9. Software Team Member Management (Section 5.22)
10. Interface with Software IV&V Agents (Section 5.23)
11. Coordination with Associate Developers (Section 5.24)
12. Improvement of Project Processes (Section 5.25)

Note: These major activities may overlap, may be applied iteratively, may be applied differently to various elements of software, and need not be performed in the order listed above.

7. The developer’s software development processes shall be described in the Software Development Plan (SDP).

Note: The sections specified in parentheses after each activity and integral process in #6 above are the SDP sections where the processes are to be described. See Section 6.2 for the SDP template identifier.

4.2 General Requirements for Software Development

This section specifies the general developer requirements for carrying out the detailed requirements in Section 5 of this standard. See Appendix H.1 for more detailed requirements.

4.2.1 Software Development Methods

1. The developer shall use systematic, documented methods for all software development activities.

2. These systematic software development methods shall be specified in the SDP.

Note: See Appendix H.1 SDP Template contents § 4.2.1 for more detailed information.

4.2.2 Standards for Software Products

1. The developer shall develop standards for representing requirements, architecture, design, code, test cases, test procedures, and test results.

2. These product representation standards shall be specified in the SDP.

3. The developer shall apply the standards specified in the SDP for representing requirements, architecture, design, code, test cases, test procedures, and test results.

Note: See Appendix H.1 SDP Template contents § 4.2.2 for more detailed information.

4.2.3 Traceability

Traceability provides a relationship between the contents of two related products, such as the relationship from each software requirement to the software design component(s) that satisfy the requirement. Bidirectional traceability provides a two-way relationship between the contents of two related products. An example of such a two-way relationship is the traceability from each software requirement to the software design component(s) that satisfy the requirement and the traceability from each design component to the software requirement(s) satisfied by the design component.

The intent of the bidirectional traceability requirements in this section is to trace both upward and downward to:

1. demonstrate that all requirements are allocated to components, implemented, and tested,

2 This subsection is based on information from Section 4.2.3 of (EIA/IEEE J-016).
2. demonstrate that no extra requirements are implemented,
3. provide information on the impacts of proposed changes,
4. provide information on the impacts of discovered problems, and
5. provide information for software maintenance.

Bidirectional traceability applies to all levels of requirements within the system, including the system level, the subsystem levels, and the software item level. At each level of the specification tree, the requirements in each parent system or subsystem specification are traced down to the applicable requirements in the child subsystem, software item, or hardware item specifications immediately beneath their parents in the specification tree, and each requirement in a child specification is traced back up to the applicable requirements in the parent specification. When requirements are derived in whole or in part from design decisions, those requirements are bidirectionally traced to the documented design decisions.

The term “subsystem” as defined in Section 1.2.5.1 is used for any number of levels between the system level and the software level. When this standard uses the word “subsystem,” it indicates one of several possible levels between the system and software levels.

As an example, consider a system that has a system specification and its design has three subsystems. The system requirements are allocated to the child subsystems, and the system requirements are traced to the child subsystem requirements. In turn, the child subsystem requirements are traced back to the parent system requirements. This process is repeated for every level in the specification tree for the system. Eventually, at the lowest level of the subsystems in the specification tree, the subsystem requirements are allocated to software items, hardware items, and manual operations. The software requirements for each software item are traced back up to its parent subsystem requirements, and the parent subsystem requirements are traced down to the software requirements. The parent subsystem for a software item is the subsystem immediately above the software item in the specification tree.

The same philosophy of bidirectional traceability applies to tracing the allocation of requirements up and down throughout the lifecycle sequence of software products. Thus, there is bidirectional traceability between software requirements and:
1. the architecture components that implement the requirements;
2. the units in the software design that implement the requirements;
3. test cases for software unit testing;
4. test cases for unit integration testing;
5. test cases for software-hardware integration testing;
6. tests in the Software Test Plan (STP);
7. test cases in the Software Test Description (STD); and
8. test procedure steps in the STD.

Bidirectional traceability also applies between other software products. For example, there is bidirectional traceability between:
1. software architecture components and the units in the software design, and
2. software units and the source code files that implement the software units.

Bidirectional traceability thus ensures that if a change is proposed to a system, subsystem, or software requirement, the impact of that change can more easily be determined. If a problem or discrepancy is detected in software, the bidirectional traceability can be used to find the related upstream products so that they can be analyzed to determine the source of the error or ambiguity. The bidirectional traceability can then be used to determine other downstream products to which the error or ambiguity could have propagated.

For efficiently producing correct bidirectional traceability information and updating that information across the lifecycle as new products are developed and changes are made, the developer is strongly
encouraged to use one or more tools for recording and managing the bidirectional traceability information. The products whose DIDs require traceability information can reference the traceability information in these tools.

1. The bidirectional traceability information specified in this section, Section 4.2.3 and its subparagraphs, shall be recorded as specified in the SDP.

### 4.2.3.1 Bidirectional Traceability for Requirements

1. The developer shall participate in defining and recording bidirectional traceability between all levels of system and subsystem requirements, including functional and nonfunctional requirements, as applicable, including all items in the traceability section of the System/Subsystem Specification (SSS) DID.

   Note: This requirement applies to all levels of requirements in the specification tree above the level of software requirements.

2. The developer shall participate in defining and recording bidirectional traceability between the system or subsystem components and system or subsystem requirements, including functional and nonfunctional requirements, that are allocated to those system or subsystem components, including all items in the traceability section of the System/Subsystem Design Description (SSDD) DID.

3. The developer shall define and record bidirectional traceability between each software requirement and the system or subsystem requirements that the software requirement addresses, including all items in the traceability section of the Software Requirements Specification (SRS) DID.

4. When requirements are derived in whole or in part from architecture or design decisions or user’s operational concepts, then those requirements shall be bidirectionally traced to the documentation of those decisions and operational concepts.

5. The developer shall define and record bidirectional traceability between each software interface requirement and the system or subsystem requirements that the software interface requirement addresses, including all items in the traceability section of the Interface Requirements Specification (IRS) DID.

### 4.2.3.2 Bidirectional Traceability for Architecture and Design

1. The developer shall define and record bidirectional traceability between the following pairs, including all items in the traceability sections of the Software Architecture Description (SAD) template:
   a. software architecture components and the software requirements and software interface requirements allocated to the architecture components; and
   b. software use cases, or equivalent, and the software requirements and software interface requirements allocated to the use cases, or equivalent.

   Note: See Section 6.2 for the SAD template identifier.

2. The developer shall define and record bidirectional traceability between the following pairs, including all items in the traceability section of the Software Design Description (SDD) DID.
   a. each software unit and each software requirement that is allocated to the software unit, and
   b. each software unit and each interface requirement that is allocated to the software unit.

   Note: The software requirements are documented in the SRS and the software interface requirements are documented in the IRS.

3. The developer shall define and record bidirectional traceability between software interfaces described in the Interface Design Description (IDD) and the system, subsystem, software, or software interface requirements allocated to that interface, including all items in the traceability section of the IDD DID.
Note: For systems with multiple levels of subsystem requirements, an interface could be defined at any level of the specification tree. Requirements for an interface could be specified in system or subsystem specifications, as applicable, or could be specified in software requirements, software interface requirements specifications, or interface control documents (ICDs).

4. The developer shall define and record bidirectional traceability, including all items in the traceability section of the Database Design Description (DBDD) DID between the:
   a. databases described in the DBDD and the system, subsystem, software, and software interface requirements allocated to those databases; and
   b. software units described in the DBDD and the system, subsystem, software, and software interface requirements allocated to those software units.

Note: For systems with multiple levels of system requirements, a database could be defined at any level of the specification tree. Requirements for a database could be specified in system or subsystem specifications, as applicable, or could be specified in software requirements specifications, including software interface requirements.

4.2.3.3 Bidirectional Traceability for Testing

1. The developer shall define and record the bidirectional traceability between each software unit test case and the software requirements and software interface requirements addressed by the software unit test case.

2. The developer shall define and record the bidirectional traceability between each unit integration and testing (I&T) test case and the software requirements and software interface requirements addressed by the unit I&T test case.

3. The developer shall define and record the bidirectional traceability, including all items in the traceability section of the Software Test Plan (STP) DID between the tests defined in the STP and the software requirements and software interface requirements addressed by those tests.

4. For software-only systems, the developer shall define and record the bidirectional traceability between the tests defined in the STP and the system or subsystem requirements addressed by those tests, including all items in the traceability section of the STP DID.

5. For each test case defined in the Software Test Description (STD), the developer shall define and record the bidirectional traceability, including all items in the traceability section of the STD DID concerning test cases between the test case defined in the STD and the software requirements and software interface requirements addressed by the test case.

6. For software-only systems, for each test case defined in the STD, the developer shall define and record the bidirectional traceability between the test case defined in the STD and the system or subsystem requirements addressed by the test case, including all items in the traceability section of the STD DID concerning test cases.

7. For each test case defined in the STD that addresses multiple requirements, the developer shall define and record the bidirectional traceability between the requirements addressed by that test case and the steps in the test procedure(s) for the test case where the requirements are addressed, including all items in the traceability section of the STD DID concerning test procedures.

8. The developer shall participate in defining and recording the bidirectional traceability between each software-hardware I&T test case and the requirements addressed by that test case.

9. For software-only systems, for each software-hardware I&T test case that addresses multiple requirements, the developer shall define and record the bidirectional traceability between the requirements addressed by that test case and the steps in the test procedure(s) for the test case where the requirements are addressed.

Note: In software-only systems, the hardware is provided to the contract, and the developer is not responsible for developing the hardware. The software-only system must interface with, or possibly control, the hardware.
10. For software-only systems, the developer shall define and record the bidirectional traceability between the system qualification test cases and the requirements addressed by those test cases.

11. For software-only systems, for each system qualification test case that addresses multiple requirements, the developer shall participate in defining and recording the bidirectional traceability between the requirements addressed by that test case and the steps in the test procedure for the test case where the requirements are addressed.

4.2.3.4 Bidirectional Traceability for Delivery to Operations and Maintenance

1. The developer shall define and record the bidirectional traceability, including all items in the traceability section of the Software Product Specification (SPS) DID:
   a. between each software unit and the source code files that implement that software unit, and
   b. between the computer hardware resource utilization measurements and the software item requirements concerning them.

Note 1: The source code files could contain newly developed code, unmodified reusable code, and modified reusable code. Here, reusable code encompasses any type of reuse source code (e.g., from legacy systems and open source software).

Note 2: The developer is encouraged to define and record the bidirectional traceability between these items during implementation and integration and testing rather than delaying until the SPS is prepared.

4.2.4 Reusable Software Products

1. The developer shall identify reusable software products for use in fulfilling the requirements of the contract.

2. The scope of the reusable software product search shall be specified in the SDP.

3. The developer shall evaluate the identified reusable software products for use in fulfilling the requirements of the contract.

4. The criteria to be used for evaluation of the reusable software product shall be as specified in the SDP.

5. The developer shall use, as a minimum, the evaluation criteria in Appendix B for selecting reusable software products.

   Note: The evaluation criteria may be weighted differently for different categories of software and for different critical requirements.

6. The selected reusable software products shall meet the data rights requirements in the contract.

7. If COTS or any other reusable software has undesired functionality that has not been removed, the developer shall demonstrate that this functionality does not impact the software or system.

Note 1: The developer can be required by the contract to develop software products specifically for reuse.

Note 2: Any requirement that calls for the development of a software product can be met by a reusable software product that fulfills the requirement and meets the criteria established in the SDP. The reusable software product can be used as-is or modified and can be used to satisfy the entire requirement or part of the requirement.

4.2.5 Assurance of Critical Requirements

1. The developer shall identify, develop, and record strategies for the following types of critical requirements:
   a. Safety;
   b. Security;
   c. Privacy protection;
   d. Reliability, maintainability, and availability;
   e. Dependability;
   f. Human Systems Integration, including human factors engineering; and
g. Mission critical functional and performance requirements as agreed to by the acquirer and developer (e.g., requirements derived from Key Performance Parameters (KPPs), accuracy requirements, and timing requirements).

2. The developer shall:
   a. identify those computer hardware and software items or portions thereof whose failure could lead to violations of the critical requirements;
   b. develop a strategy, including the development approaches, tests, modeling, and analyses, to ensure that the requirements, architecture, design, implementation, and operating procedures for the identified computer hardware and software minimize or eliminate the potential for such violations;
   c. record the strategy in the SDP;
   d. implement the strategy; and
   e. produce evidence, as part of the required computer hardware and software products, that the strategy has been successfully carried out.

4.2.6 Computer Hardware Resource Utilization

1. The developer shall analyze requirements and design constraints concerning computer hardware resource utilization (such as maximum allowable use of processor capacity, memory capacity, input and output device capacity, auxiliary storage device capacity, and communications and network bandwidth capacity).

2. In order to meet the computer hardware resource utilization requirements and design constraints, the developer shall:
   a. allocate requirements and design constraints to computer hardware resources,
   b. model the utilization of these resources based on the software architecture and design,
   c. monitor the utilization of these resources during implementation, integration and testing, and qualification testing, and
   d. reallocate resources or identify the need for additional resources as necessary.

   Note: All of these activities always include recording the information and results even if that is not explicitly stated.

3. The allocated resource utilizations shall be managed as Technical Performance Measures (TPMs).

   Note: The computer hardware resource utilization examples cited above might not include the complete set of software TPMs.

4.2.7 Recording Rationale

1. The developer shall record rationale for key decisions made in specifying, architecting, designing, implementing, testing, and deploying the software.

2. The rationale shall include tradeoffs considered, analysis methods, and criteria used to make the decisions.

3. The rationale shall be recorded in documents, development tools, code comments, or other media that will transition to the maintenance organization, whether contractor or acquirer.

4. The meaning of “key decisions” and the approach for providing the rationale shall be described in the SDP.

4.2.8 Access for Acquirer Review

1. The developer shall provide the acquirer access to all software team member facilities, including the software engineering environment and software integration and qualification test environment, for review of software products and activities.

2. The developer shall provide the acquirer electronic access to the draft and final work products specified in Section 4 and Section 5 produced by the developer and all software team members in accordance with the contractual requirements.
3. The developer shall provide the acquirer electronic access to the draft and final work products together with tools to remotely query, report, and display that data.
Note: These requirements do not negate security policies that apply.

4.2.9 Contractual Requirements for Software
1. The developer shall plan the approach to be followed for meeting all of the contractual requirements regarding software.
2. The developer shall record in the SDP the approach to be followed for meeting all of the contractual requirements.
5. Detailed Requirements

The order of the requirements in this section is not intended to specify the order in which they must be carried out. Some activities might not be applicable to the project or contracted effort. Many of the activities can be ongoing at the same time; different software products can proceed at different paces; and activities specified in early subsections can depend on input from activities in later subsections. If the software is developed in multiple builds, some activities might be performed in every build, others might be performed only in selected builds, and activities and software products might not be complete until several or all builds are accomplished. Nonmandatory notes throughout Section 5 explain how to interpret each activity on a project involving multiple builds. A project involving a single build accomplishes all required activities in that build.

Note: The wording here and throughout this standard is intended to:
1. emphasize that the development and recording of planning and engineering information is an intrinsic part of the software development process, to be performed regardless of whether a deliverable product is required;
2. use the Data Item Description (DID) or template as the minimum list of items to be covered in the planning or engineering activity regardless of whether a CDRL item delivery is required for the DID or template or whether the DID or template has been tailored; and
3. permit representations other than traditional documents for recording the information (e.g., computer-aided engineering (CAE) tools).

5.1 Project Planning and Oversight

This section specifies the developer requirements for project planning and oversight.

Note: If a system or software item is developed in multiple builds, planning for each build is interpreted to include:
1. overall planning for the contract,
2. detailed planning for the current build, and
3. planning for future builds to a level of detail compatible with the information available.

5.1.1 Software Development Planning

1. The developer shall develop and record plans and approaches for conducting the activities required by this standard.
2. The developer shall develop and record plans and approaches for conducting the activities required by other software-related requirements in the contract.
3. The planning for the activities required by this standard shall be consistent with system-level planning.
4. The planning for the activities required by this standard and by the contract shall be documented in the Software Development Plan (SDP).
5. The planning for the activities required by this standard shall include all items in the SDP template in Appendix H.1 Software Development Plan Template.
6. The Software Development Plan shall be prepared in accordance with the SDP template in Appendix H.1 Software Development Plan Template.
7. The SDP shall be an integrated plan covering the software development activities for all software team members throughout development.
8. The developer shall define and record in the SDP the artifacts and other information to be developed and recorded to satisfy the contents of the software products required by this standard.
9. The developer shall define and record in the SDP the artifacts and other information to be developed and recorded to prepare all software-related deliverable products required by the contract.

10. The developer shall use the DID or template for a product as the minimum list of items to be covered in the planning or engineering activity regardless of whether a CDRL item delivery is required for the DID or template or whether the DID or template has been tailored.

5.1.2 Software Integration and Qualification Test Planning

5.1.2.1 Software Integration Planning

This section pertains to unit integration and software-hardware item integration.

1. The developer shall develop and record the planned sequence of integration of the:
   a. software units,
   b. software items, and
   c. software items with the hardware items on which they execute.

2. The planned integration sequence of software units, software items, and hardware items shall be recorded as part of the Software Master Build Plan (SMBP).

3. The software integration planning shall include all applicable items in Appendix H.3 Software Master Build Plan (SMBP) Template.
   Note: See Section 6.2 for the SMBP template identifier.

4. The developer shall record the integration and test planning for software units in the SDFs.

5.1.2.2 Software Item Qualification Test Planning

1. The developer shall develop and record plans for conducting software item qualification testing.

2. The plans for conducting software item qualification testing shall be documented in the Software Test Plan (STP).
   Note: See Section 6.2 for the STP DID identifier.

3. The software item qualification test planning shall include all applicable items in the Software Test Plan (STP) DID, as defined in the SDP (see Section 5.1.1).

4. The software item qualification test planning shall address all verification methods, i.e., Inspection (I), Analysis (A), Demonstration (D), and Test (T), necessary to fully verify the software requirements, including the software interface requirements.

5. The software item qualification test planning shall address all verification levels in the verification testing hierarchy (e.g., build, software item, subsystem, system) necessary to fully verify the software requirements, including software interface requirements.

6. The developer shall plan for recording and tracking the cumulative record of the verification status, i.e., fully verified, partially verified, not verified, of all software requirements, including software interface requirements, for:
   a. all verification testing, i.e., initial qualification test execution, regression testing, and retesting,
   b. all test cases for all verification methods, i.e., Inspection (I), Analysis (A), Demonstration (D), and Test (T); and
   c. all levels in the verification testing hierarchy, i.e., build, software item, subsystem, system.

7. The software item qualification test planning shall include criteria for:
   a. test activity suspension,
   b. test activity termination, and
   c. test activity resumption after suspension and termination (e.g., test cases, test procedures, and software are corrected).
5.1.3 System Qualification Test Planning

1. The developer shall participate in developing and recording plans for conducting system qualification testing.

2. The system qualification test planning shall address all verification methods, i.e., Inspection (I), Analysis (A), Demonstration (D), and Test (T), necessary to fully verify the system requirements, including the system interface requirements.

3. The system qualification test planning shall address all verification levels in the verification testing hierarchy (e.g., build, item, subsystem, system) necessary to fully verify the system requirements, including system interface requirements.

Note: “System qualification testing” in this section is interpreted to mean the verification of all levels of requirements higher in the specification tree than the software requirements (e.g., subsystem, system).

5.1.4 Planning for Software Transition to Operations

1. The developer shall develop and record plans for transitioning software to operations.

2. The developer shall specify which software is to be fielded, i.e., distributed, to which user sites. See Section 5.12.4.

3. The planning for transition to operations shall include plans for performing:
   a. software installation,
   b. software and configuration checkout at the user sites, and
   c. training at the user sites specified in the contract.

4. The developer shall plan for transitions to operations of software deliveries, including processes to ensure:
   a. uninterrupted operations through the transition, with a planned down time, if any, acceptable to the acquirer and users for transition; and
   b. continuity of operational data through each transition.

5. The planning for transition to operations shall include all applicable items in the Software Installation Plan (SIP) DID, as defined in the SDP (see Section 5.1.1).

6. The software transition planning and execution shall be consistent with the system transition planning and execution.

   Note: See Section 6.2 for the SIP DID identifier.

Note 1: If software is developed in multiple builds, the developer’s build planning identifies for each build what software, if any, is to be fielded, i.e., distributed, to users and the extent of fielding (for example, full fielding or fielding to selected evaluators only). Preparing for software use for each build is interpreted to include those activities necessary to carry out the fielding plans for that build.

Note 2: The standard does not cover hardware installation.

5.1.5 Planning for Software Transition to Maintenance

1. The developer shall develop and record plans:
   a. identifying the software development resources that will be needed by the maintenance organization to fulfill the support concept specified in the contract, and
   b. describing the approach to be followed for transitioning deliverable items to the maintenance organization.

2. The developer shall plan for:
   a. software engineering and software integration and qualification test environments at the maintenance site, and
   b. maintaining the software engineering and software integration and qualification test environments after development to support discrepancy resolution and changes to software and operational constants.
3. The planning for transition to maintenance shall include all applicable items in the Software Transition Plan (STrP) DID, as defined in the SDP (see Section 5.1.1).

Note: See Section 6.2 for the STrP DID identifier.

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

5.1.6 Following and Updating Plans
1. The developer shall conduct the software development activities in accordance with the plans in Section 5.1.
2. All software team members shall conduct the software development activities in accordance with the plans in Section 5.1.
3. The developer shall update the plans in Section 5.1 and any other plans:
   a. whenever changes occur, and
   b. according to the requirements in the contract.
4. The plans in Section 5.1, including any updates to those plans, shall be subject to approval according to developer processes and, if required in the contract, acquirer approval.
5. The developer shall notify the acquirer when changes occur to:
   a. plans in Section 5.1, and
   b. the documents and processes referenced in those plans.

5.2 Establishing a Software Development Environment
This section specifies the requirements for the software development environment. The software development environment consists of both the software engineering environment and the software integration and qualification test environment.

1. The developer shall establish and maintain a software development environment that provides the resources necessary to perform all activities specified in this standard.
2. In order to ensure that the resources are sufficient to accomplish the software work required by this standard, the developer shall analyze the adequacy of the planned:
   a. software engineering environment, and
   b. software integration and qualification test environment.
3. The developer shall record the results of the adequacy assessments for the software engineering environment and software integration and qualification test environment in the SDP.

Note: If a system or software item is developed in multiple builds, establishing the software development environment in each build is interpreted to mean establishing the environment needed to complete that build.

5.2.1 Software Engineering Environment
1. The developer shall establish, control, and maintain a software engineering environment to perform the software engineering effort.
2. The developer shall keep the tools, including COTS and open source tools, in the software engineering environment used to produce the software current.
3. The developer shall demonstrate that each element of the software engineering environment performs its intended functions.
4. The developer shall demonstrate that each element of the software engineering environment is ready to perform its intended functions before starting the activity that uses that element.
5.2.2 Software Integration and Qualification Test Environment
1. The developer shall establish, control, and maintain a software integration and qualification test environment to perform integration and qualification testing of software.
2. The developer shall demonstrate that the tools, including COTS and open source tools, in the software integration and qualification test environment are kept current.
3. The developer shall demonstrate to the acquirer that each element of the software integration and qualification test environment performs its intended functions.
4. The developer shall validate each element of the software integration and qualification test environment that is used in verification of requirements before the elements are used for that purpose.
5. The developer shall demonstrate that each element of the software integration and qualification test environment is ready to perform its intended functions before starting the integration or qualification testing activity that uses that element.

5.2.3 Software Development Library
1. The developer shall establish, control, and maintain a software development library (SDL) to facilitate the orderly development and subsequent maintenance of software.
   Note: The SDL may be an integral part of the software engineering environment and software integration and qualification test environment.
2. The developer shall maintain the SDL throughout the system development lifecycle.

5.2.4 Software Development Files
1. The developer shall establish, control, and maintain a software development file (SDF):
   a. for each software unit or logically related group of software units,
   b. for each software item,
   c. for each build,
   d. for logical groups of software items, as applicable,
   e. for subsystems, if applicable, and
   f. for the overall system.
2. The developer shall record information about the development of the software in appropriate SDFs.
3. The developer shall maintain the SDFs throughout the system development lifecycle.

5.2.5 Nondeliverable Software
1. The developer shall demonstrate that all nondeliverable software products used perform their intended functions.
2. If any nondeliverable software product is used in the development of deliverable software products, then the developer shall demonstrate that:
   a. after delivery to the acquirer, the operation and maintenance of the deliverable software products do not depend on the nondeliverable software products, or
   b. the acquirer has or can obtain the same nondeliverable software products.
3. If any nondeliverable software product is used in the development of deliverable software products, then the developer shall demonstrate that the nondeliverable software has caused no adverse effects to the deliverable software.
   Note: Examples of these adverse effect include, e.g., corruption of deliverable software or data, introduction of viruses, Trojan horses, or other malware.

5.3 System Requirements Analysis
This section specifies the developer requirements for participation in system requirements analysis.
Note: If a system is developed in multiple builds, it is possible that its requirements are not fully defined until the final build. The developer's planning identifies the subset of system requirements to be defined in each build and the subset to be implemented in each build. System requirements analysis for a given build is interpreted to mean defining the system requirements so identified for that build.

5.3.1 Analysis of User Input

1. The developer shall participate in analyzing user input provided by the acquirer to gain an understanding of the user needs.
   Note 1: This user input takes the form of need statements, surveys, discrepancy and change reports, feedback on prototypes, interviews, or other user input or feedback.
   Note 2: This requirement is intended to ensure that all interested parties maintain ongoing communications regarding user needs throughout the development of the system. Interested parties include, but are not limited to, the users, operators, acquirer, developer, maintenance organizations, and test organizations.

5.3.2 Operational Concept

1. The developer shall participate in defining and recording the operational concept for the system.
2. The results of defining and recording the operational concept shall include all applicable items in the Operational Concept Description (OCD) DID, as defined in the SDP (see Section 5.1.1).
   Note: See Section 6.2 for the OCD DID identifier.

5.3.3 System Requirements Definition

1. Based on the analysis of user needs, the operational concepts, and other considerations, the developer shall participate in defining and recording the requirements, including derived requirements, to be met by the system.
2. The developer shall participate in defining and recording the system interface requirements.
   Note: An interface could be specified in system or subsystem specifications, as applicable, or could be specified in interface requirements specifications or interface control documents (ICDs).
3. The developer shall participate in defining and recording the methods to be used for verifying that each system requirement, including interface requirements, has been met.
4. The results of defining and recording the requirements and verification methods shall include all applicable items in the System/Subsystem Specification (SSS) DID, as defined in the SDP (see Section 5.1.1).
   Note: See Section 6.2 for the SSS DID identifier.
5. The developer shall participate in the allocation of system requirements to hardware, software, and manual operations.
   Note 1: System requirements include both functional and nonfunctional requirements.
   Note 2: If a system consists of subsystems, the activity in Section 5.3.3 is intended to be performed iteratively with the activities in Section 5.4 (System Architecture and Design) to define system requirements; architect and design the system, identifying its subsystems; allocate the system requirements to the identified subsystems; define the requirements for those subsystems; architect and design the subsystems, identifying their components; and so on until, in the last iteration, requirements are allocated to hardware items, software items, and manual operations.

5.4 System Architecture and Design

This section specifies the developer requirements for participation in system architecture and design.

---

3 Note 2 is based on information from Section 5.3.1 of (EIA/IEEE J-016).
Note: If a system is developed in multiple builds, it is possible that its architectural design is not fully defined until the final build. The developer’s planning identifies the portion of the system architectural design to be defined in each build. System architectural design for a given build is interpreted to mean defining the portion of the system architecture and design identified for that build.

### 5.4.1 System-wide Architectural Design Decisions

1. The developer shall participate in defining and recording system-wide architectural design decisions, that is, decisions about the system’s behavioral design and other decisions affecting the selection and design of system components to meet both functional and nonfunctional requirements.

   Note: Architectural design decisions remain at the discretion of the developer unless converted to requirements. The developer is responsible for fulfilling all requirements and demonstrating this fulfillment through qualification testing (see Section 5.9 and Section 5.11). Architectural design decisions act as developer-internal “requirements,” to be implemented, imposed on other software team members, if applicable, and confirmed by unit, unit integration, or software-hardware integration testing, but their fulfillment need not be demonstrated to the acquirer. (See Section 5.7, Section 5.8, and Section 5.10.)

2. The result of defining and recording system-wide architectural design decisions shall include all applicable items in the systemwide design decisions section of the System/Subsystem Design Description (SSDD) DID, as defined in the SDP (see Section 5.1.1).

   Note: See Section 6.2 for the SSDD DID identifier.

### 5.4.2 System Architectural Design

1. The developer shall participate in defining and recording the system architectural design.

2. The developer shall participate in selecting software items, as part of defining the system architectural design.

3. The developer shall participate in defining:
   a. the architectural approaches that were considered to address the critical and nonfunctional requirements, and
   b. the rationale for the selected approach.

4. The developer shall participate in identifying the system architectural design risks for the selected architectural design.

5. The resulting system architecture shall include all applicable items in the system architectural design section of the SSDD DID, as defined in the SDP (see Section 5.1.1).

   Note 1: For conciseness, this subsection is written in terms of organizing a system into components. However, this is interpreted to cover organizing a system into subsystems; organizing a subsystem into hardware items, software items, and manual operations; or other variations as applicable.

   Note 2: See Section 6.2 for the SSDD DID identifier.

   Note 3: See Section 4.2.3.1 for the traceability requirements between system requirements and system architectural design components.

### 5.5 Software Requirements Analysis

This section specifies the developer requirements for software requirements analysis.

1. Based on the analysis of system requirements, the system architecture and design, the operational concept, and other considerations, the developer shall define and record for each software item:

---

4 This subsection is based on information from Section 5.4.2 of (EIA/IEEE J-016).
a. the software requirements, including functional and nonfunctional requirements, to be met by each software item, and
b. the methods and levels for verifying each requirement.
Note: See Section 4.2.3.1 for traceability requirements for software requirements.

2. The specification of software requirements shall include requirements derived from trade studies, architecture, design decisions, and operational concepts, which might not be directly traceable to higher-level requirements.

3. The specification of each derived software requirement shall include the traceability and rationale explaining the relationship to the trade studies, system architecture, software architecture, design decisions, and operational concepts that led to the derived requirement.
Note: See Section 4.2.3.1 for traceability requirements for derived software requirements.

4. The developer shall identify the critical software requirements for the critical requirements categories specified in Section 4.2.5.

5. The software requirements analysis result shall include all applicable items in the Software Requirements Specification (SRS) DID, as defined in the SDP (see Section 5.1.1).

6. The software requirements analysis result shall include all applicable items in the Interface Requirements Specification (IRS) DID, as defined in the SDP (see Section 5.1.1).
Note 1: If a software item is developed in multiple builds, its requirements might not be fully defined until the final build. The developer’s planning identifies the subset of each software item’s requirements to be defined in each build and the subset to be implemented in each build. Software requirements analysis for a given build is interpreted to mean defining the software item requirements identified for that build.
Note 2: See Section 6.2 for the SRS and IRS DID identifiers.
Note 3: If a separate IRS is not used, the software interface requirements are specified in SRS Paragraph 3.3 and its subparagraphs.

5.6 Software Architecture and Design
This section specifies the developer requirements for software architecture and design.
Note: If a software item is developed in multiple builds, its architecture and design might not be fully defined until the final build. Software architecture and design in each build is interpreted to mean the architecture and design necessary to meet the software item requirements to be implemented in that build.

5.6.1 Overall Software Architecture
1. The developer shall define and record the software architecture of the complete set of software under the contract, including all categories of software.
2. The resulting overall software architecture shall include all applicable items in Paragraphs 3 through 5 of the Software Architecture Description (SAD) template, as defined in the SDP (see Section 5.1.1).
Note 1: See Section 4.2.3.2 for traceability requirements for the software architecture.
Note 2: See Section 6.2 for the SAD template identifier.
3. The developer shall update the Software Master Build Plan (SMBP) to reflect the software architecture and its impact on integrating the software items.
Note: See Section 6.2 for the SMBP template identifier.

5.6.2 Software Item Architecture
1. The developer shall define and record the software architecture of each software item.
2. The resulting software architecture for each software item shall include all applicable items in Paragraphs 3, 4, and 6 of the Software Architecture Description (SAD) template, as defined in the SDP (see Section 5.1.1).
Note 1: See Section 6.2 for the SAD template identifier.
Note 2: See Section 4.2.3.2 for the traceability requirements for software architecture.

3. The developer shall update the Software Master Build Plan (SMBP) to reflect the software architecture and its impact on integrating the software units and software items.
Note: See Section 6.2 for the SMBP template identifier.

5.6.3 Software Item Detailed Design

1. The developer shall develop and record the detailed design of each software item.
2. The detailed design of each software item shall include all applicable items in Sections 3 and 4 of the Software Design Description (SDD) DID that are not included in the SAD, as defined in the SDP (see Section 5.1.1).
3. The detailed design of each software item shall include all applicable items in Section 5 of the Software Design Description (SDD) DID, as defined in the SDP (see Section 5.1.1).
Note 1: See Section 6.2 for the SDD DID identifier.
Note 2: See Section 4.2.3.2 for the traceability requirements for each software unit covered by the SDD.
4. The detailed design of the software item interfaces shall include all items in the Interface Design Description (IDD) DID, as defined in the SDP (see Section 5.1.1).
Note 1: See Section 6.2 for the IDD DID identifier.
Note 2: If a separate IDD is not used, the software interface design are specified in SDD Paragraph 4.3 and its subparagraphs.
Note 3: See Section 4.2.3.2 for the traceability requirements between interfaces in the IDD and software item requirements that address the interface.
5. If the software item contains a user interface, the detailed design of the software item shall include the design of the user interface screens and interaction mechanisms, as defined in the SDP (see Section 5.1.1).
Note: See Section 4.3 of the SDD DID.
6. The detailed design of the software units that are databases or are software units that access or manipulate databases shall include all items in the Database Design Description (DBDD) DID, as defined in the SDP (see Section 5.1.1).
Note 1: See Section 6.2 for the DBDD DID identifier.
Note 2: See Section 4.2.3.2 for the traceability requirements for each database or other software unit covered by the DBDD.
7. The developer shall record the software detailed design information in the applicable software development files (SDFs).
8. If changes are needed in automatically generated code, the developer shall modify the design in the tool that generated the code, rather than modifying the automatically generated code.
Note: The need for change is identified in a Discrepancy and Change Request (DCR).

5.7 Software Implementation and Unit Testing

This section specifies the developer requirements for software implementation and software unit testing. Software implementation activities include such tasks as coding computer instructions, coding data definitions, building databases, populating databases with data values, creating procedures, populating other data files with values, configuring COTS and any other reusable software, modifying existing code and data, employing automated code generators, and many other tasks needed to implement the design. Software units in the design do not necessarily have a one-to-one relationship with the code and data entities (e.g., classes, objects, procedures, databases, data files) that implement them or with the computer files containing those entities.
Unit testing is performed to find any discrepancies that escaped peer reviews and to determine whether the software unit meets all of its allocated requirements and implements its design for both nominal and off-nominal conditions. Unit testing is accomplished for each software unit and includes preparing for testing, the unit, performing the unit testing on the software unit, analyzing and recording the test results, fixing problems, and performing retesting. Requirements are also included in this section for regression testing the software unit if changes (e.g., to the software or the target computer system) are made after the unit has successfully completed unit testing in order to determine whether the changes adversely impact the functioning or performance of the unit. Unit testing is an iterative process of initial testing, fixing problems in the software or test preparation products (e.g., test data, test procedures), and retesting until the unit test cases, including the nominal and off-nominal test cases, all execute successfully, i.e., with actual results matching expected results.

Unit testing is usually performed on one software unit at a time in isolation. The definition of software unit allows for a hierarchy of units, hence some units may need to be tested with other units present.

If a software item is developed in multiple builds, software implementation and unit testing of that software item will not be completed until the final build. Software implementation and unit testing for each build is interpreted to include those software units, or parts of software units, needed to meet the software item requirements to be implemented in that build.

The terms “unit test” and “unit testing” refer to the activity of verifying the correctness of the software unit implementation. The use of the words “test” or “testing” in this section is distinct from the verification method of Test. The activity of unit testing can require the use of all verification methods, i.e., Inspection (I), Analysis (A), Demonstration (D), and Test (T).

5.7.1 Implementing Software

1. The developer shall implement and record software corresponding to each software unit in the software item design.
2. The developer shall record the software implementation products in the appropriate software development files (SDFs).

5.7.2 Preparing for Unit Testing

1. The developer shall prepare to unit test all newly developed software.
2. The developer shall prepare to unit test all of the following reusable software within the software unit for which source code or design is available:
   a. modified reusable software;
   b. reusable software where previous discrepancy and change reports indicate potential problems, even if the reusable software has not been modified;
   c. reusable software whose unit testing did not satisfy the requirements of this section, i.e., Section 5.7.2, of this standard;
   d. reusable software partially or fully satisfying a critical requirement (see Section 4.2.5), even if the reusable software has not been modified; and
   e. reusable software when the adequacy of the unit testing is unknown, i.e., whether the requirements of this section, i.e., Section 5.7.2, were satisfied is unknown.

   Note: If neither the source code nor the design is available for reusable software in the software unit, then unit testing with respect to the design is not required for that part of the unit.
3. The developer shall develop the following test preparation products for testing the software corresponding to each software unit:
   a. instructions for preparing the test environment;
   b. test cases for all verification methods used for testing the software units;
   c. test procedures;
d. test drivers or test scripts, or both; and
e. test data, including test databases.

Note: See Section 4.2.3.3 for traceability requirements for software unit test cases.

4. For each software unit the developer shall define unit test equivalence classes and representative sets of nominal and off-nominal conditions, including, as a minimum, valid and invalid values; values just inside the boundary of acceptable range values, at the boundary, and just outside the boundary; and extreme values.

5. The developer shall base the unit test equivalence classes on:
   a. the unit’s source code for all newly developed software, unless the source code was automatically generated;
   b. the unit’s design for all automatically generated code in the unit, whether newly generated or reused;
   c. the unit’s code for all reusable software where source code is available, unless it was automatically generated; and
   d. the unit’s design for all reusable software where the source code is not available.

6. Using these representative sets of nominal and off-nominal conditions, the developer shall define unit test cases that address the unit’s design, including, as a minimum, correct execution of:
   a. all algorithms;
   b. all interfaces internal to the unit, including interfaces to reusable software within the unit;
   c. all interfaces external to the unit;
   d. all statements and branches;
   e. concurrent access of the same data, i.e., reading, adding, updating, and deleting data by multiple users or systems;
   f. all error and exception handling within the unit;
   g. all fault detection, isolation, recovery (e.g., failover), and data capture and reporting; and
   h. all startup, termination, and restart conditions, when applicable.

Note: To satisfy the above requirements for automatically generated code, it is strongly recommended that the developer make maximum use of the tools provided in the software development environment for analyzing and exercising the automatically generated code.

7. Using these representative sets of nominal and off-nominal conditions, the developer shall define unit test cases that address the unit’s requirements, including, as a minimum, all:
   a. software requirements, or the portions thereof, allocated to the unit; and
   b. software interface requirements, or the portions thereof, allocated to the unit.

8. The developer shall define at least one unit test case for each unit test equivalence class of nominal and off-nominal conditions.

9. The developer should automate the preparation and execution of the unit test cases, test procedures, test drivers, test scripts, and test data to the extent feasible.

10. The developer shall record the unit test preparation products in the appropriate SDFs.

5.7.3 Performing Unit Testing

1. The developer shall perform unit testing in accordance with the unit test preparation products prepared according to the requirements in Section 5.7.2 (preparing).
2. The developer shall repeat the unit testing in accordance with Section 5.7.4 (analyzing and recording) and Section 5.7.6 (retesting) until all test cases and test procedures have been performed successfully, i.e., with actual results matching the expected results.
5.7.4 Analyzing and Recording Unit Testing Results

1. For all unit testing performed, including initial unit testing, regression testing, and retesting, the developer shall perform the following:
   a. analyze the results of unit testing, and
   b. identify all discrepancies found during unit testing.

2. As each unit test case and test procedure has been performed successfully, i.e., with actual results matching expected results, during initial unit testing, regression testing, and retesting, the developer shall:
   a. record the unit test results in the appropriate software development files (SDFs), and
   b. record the unit test analysis results in the appropriate SDFs.

5.7.5 Unit Regression Testing

1. For each software unit, the unit regression test suite shall consist of all unit test preparation products for that software unit, including:
   a. instructions for preparing the test environment;
   b. test cases for all verification methods used in regression testing the software units;
   c. test procedures;
   d. test drivers or test scripts, or both; and
   e. test data, including test databases.

2. The developer shall update the unit regression test suite to reflect new and changed requirements, including interface requirements, and design.

3. The developer shall record the unit regression test suites in the appropriate SDFs.

4. The developer shall execute the software unit regression test suite in accordance with Section 5.7.3 (performing) and Section 5.7.6 (retesting) after any changes, i.e., additions, deletions, or modifications, to:
   a. the previously tested unit,
   b. any reusable software included or integrated with the software unit,
   c. the target computer system(s), or
   d. the test environment.

5. The developer should automate the unit regression test suite to the extent feasible.

5.7.6 Revising and Retesting Units

1. Based on the results of unit testing, including initial testing, regression testing, and retesting, the developer shall make all necessary revisions to the:
   a. software;
   b. instructions for preparing the test environment;
   c. test cases for all verification methods used for retesting the units;
   d. test procedures;
   e. test drivers or test scripts, or both;
   f. test data, including test databases;
   g. regression test suite; and
   h. test environment.

2. The developer shall update the software development files (SDFs) as needed to reflect the revisions for software unit testing.

3. The developer shall update other software products as needed to reflect the revisions for software unit testing.

   Note: Other software products include, for example, software architecture, design, and user manuals.

4. The developer shall retest the software unit:
a. in accordance with Section 5.7.3 (performing) and Section 5.7.4 (analyzing and recording), and

b. in accordance with the applicable test cases and test procedures prepared according to Sections 5.7.2 (preparing), Section 5.7.5 (regression), and Section 5.7.6 (retesting).

Note: See Section 5.7.4 for the requirements for analyzing and recording the unit testing activities and results.

5.8 Unit Integration and Testing

This section specifies the developer requirements for software unit integration and testing. Unit integration and testing is performed to integrate the software corresponding to two or more software units, testing the resulting software to ensure that it works together as intended, and continuing this process until all software in each software item is integrated and tested. Unit integration and testing is performed in the unit integration sequence specified in the Software Master Build Plan (SMBP) (see Section 5.1.2).

The term “unit integration and testing” or “unit I&T” is used throughout this section to refer to the activity of verifying the correct functioning of an integrated collection of units. Unit I&T is an iterative process accomplished for each software item and includes preparing for integrating the units, preparing for testing the integrated units, integrating the units, testing the integrated units, analyzing and recording the test results, fixing problems in the software or test preparation products (e.g., test data, test procedures), and retesting until the unit I&T test cases, including all nominal and off-nominal test cases, all execute successfully, i.e., with actual results matching expected results.

Requirements are also included in this section for regression testing the integrated units if changes (e.g., to the software or the target computer system(s)) are made after the integrated units have successfully completed unit integration and testing in order to determine whether the changes adversely impact the functioning or performance of the integrated software units.

If a software item is developed in multiple builds, unit I&T of that software item will not be completed until the final build. Unit I&T for each build is interpreted to mean integrating software developed in the current build with other software developed in that and previous builds, and testing the results. Software units from more than one software item can be combined in a build.

As unit integration and testing becomes more complete, the unit integration and test environment evolves to be more and more like the operational environment. The unit integration and test environment is likely to begin on a development computer system, then possibly move to an emulator or a computer system more akin to the target computer system, and eventually to the target computer system(s).

The last stage of the unit I&T is called developer-internal software item integration testing. If the software item is developed in multiple builds, then the developer-internal software item integration testing occurs on a build-by-build basis.

Since units can consist of other units, some unit I&T can take place during unit testing. The requirements in this section are not meant to duplicate those activities.

The use of the words “test” or “testing” in this section is distinct from the verification method of Test. The activity of unit integration and testing can require the use of all verification methods, i.e., Inspection (I), Analysis (A), Demonstration (D), and Test (T).

5.8.1 Testing on the Target Computer System

The developer is strongly encouraged to meet the requirements in this section, i.e., Section 5.8.1, as early as possible in unit integration and testing (I&T). See the definition of target computer system in Section 3.1.
1. Whenever possible, the developer shall perform unit I&T on the target computer system(s) in a configuration as close as possible to the operational configuration.

2. Wherever possible, the developer shall perform the unit I&T using the actual interfaces.

3. If using actual interfaces is not possible for unit I&T, then the developer shall use high-fidelity simulations of the interfaces.

   Note: See Section 5.2.2 for validation requirements for simulations, simulators, and other software used for verification of requirements.

4. The developer shall conduct all developer-internal software item integration testing under conditions as close as possible to those that the software will encounter in the operational environment, i.e., target computer system(s), operational data constants, operational input and output data rates, operational scenarios.

5.8.2 Preparing for Unit Integration and Testing

1. The developer shall prepare to perform unit I&T on:
   a. all newly developed software, and
   b. all modified and unmodified reusable software, including COTS software.

2. The developer shall prepare for integrating the software units in accordance with the integration sequence recorded in the Software Master Build Plan (SMBP) (see Section 5.1.2).

3. The developer shall develop the following test preparation products for conducting unit I&T:
   a. instructions for preparing the test environment;
   b. test cases for all verification methods used for unit I&T;
   c. test procedures;
   d. test drivers or test scripts, or both; and
   e. test data, including test databases.

   Note: See Section 4.2.3.3 for traceability requirements for software unit I&T.

4. The developer shall define and record unit I&T equivalence classes and representative sets of nominal and off-nominal conditions, including valid and invalid values; values just inside the boundary of acceptable range values, at the boundary, and just outside the boundary; and extreme values.

5. Using these representative sets of nominal and off-nominal conditions, the developer shall define and record unit I&T cases that address the software design, including, as a minimum, correct execution of:
   a. all algorithms;
   b. all end-to-end functional capabilities through the software units under test;
   c. all software interfaces among the software units under test;
   d. all software interfaces external to the software units under test;

   Note: Testing the external interfaces early in the integration sequence is encouraged.

   e. scenarios containing multiple users or functions that must execute concurrently, using normal and heavy operational workloads;
   f. concurrent access of the same data, i.e., reading, adding, updating, and deleting data by multiple users or functions in the software units under test, using normal and heavy operational workloads;
   g. all integrated error and exception handling across the software units under test;
   h. all fault detection, isolation, recovery (e.g., failover), and data capture and reporting;
   i. all startup, termination, and restart conditions, when applicable;
   j. all relevant stress conditions, including worst-case scenarios (e.g., extreme workloads, high frequency of inputs and events, large number of users, simulated failed hardware, missing or malfunctioning interfaces, tight timelines); and
   k. endurance testing using normal and heavy operational workloads.
6. Using these representative sets of nominal and off-nominal conditions, the developer shall define unit I&T cases that address the software requirements, or portions thereof, allocated to the software units under test, including, as a minimum, all:
   a. software requirements;
   b. software interface requirements;
   c. performance requirements, including timing and accuracy requirements;
   d. computer hardware resource utilization measurement requirements (e.g., CPU, memory, storage, bandwidth); and
   e. software specialty engineering requirements (e.g., supportability, testability, dependability, reliability, maintainability, availability, safety, security, and human system integration, including human factors engineering, as applicable).

7. The unit I&T cases to be executed shall include at least one test case for each unit I&T equivalence class of nominal and off-nominal conditions.

8. The developer shall define unit I&T cases to determine whether the software under test is impacted by unrequired functionality in COTS software or any other reusable software.

9. The developer shall generate and prepare the software from the controlled configuration management system for the unit I&T environment, including:
   a. executable software;
   b. procedures, if any;
   c. data files, including databases; and
   d. other software files needed to install, operate, and test the software on its target computer system(s).

10. The developer shall automate the test cases, test procedures, test drivers, test scripts, and test data to the extent feasible.

11. The developer shall record the software unit I&T preparation results in the appropriate software development files (SDFs).

5.8.3 Performing Unit Integration and Testing

1. The developer shall integrate the units in accordance with the unit integration sequence in the Software Master Build Plan (SMBP). See Section 5.1.2.

2. The developer shall perform testing of the integrated units in accordance with the test cases and test procedures prepared according to the requirements in Section 5.8.2.

3. For all unit integration and testing performed, including initial testing, regression testing, and retesting, the developer shall record:
   a. a complete unit I&T test log as the unit integration and testing proceeds;
   b. in this test log, as a minimum, the test log information specified in Appendix F, Section F.2 for unit I&T;
   c. in this test log all test incidents found during unit I&T, whether or not they are resolved during this activity;
   d. in this test log all instances, if any, of test incidents with the same symptoms, whether or not they are resolved during this activity; and
   e. in this test log references to all discrepancy and change reports, whether or not they are resolved during this activity.

4. The developer shall repeat the unit I&T testing in accordance with Section 5.8.4 (analyzing and recording) and Section 5.8.6 (retesting) until all test cases and test procedures have been performed successfully, i.e., with actual results matching the expected results.

5.8.4 Analyzing and Recording Unit Integration and Test Results

1. For all unit integration and testing performed, including initial testing, regression testing, and
retesting, the developer shall:

a. analyze the results of unit I&T;

b. record in discrepancy and change reports all discrepancies found during unit I&T, whether or not they are resolved during this activity;

c. record in each discrepancy and change report, as a minimum, the discrepancy and change report information specified in Appendix C, Section C.2 for unit I&T;

d. record the unit I&T results in the appropriate SDFs; and

e. record the unit I&T analysis results in the appropriate SDFs.

5.8.5 Unit Integration Regression Testing

1. The developer shall define a unit I&T regression test suite, including:

a. instructions for preparing the test environment;

b. test cases for all verification methods used for unit I&T regression testing;

c. test procedures;

d. test drivers or test scripts, or both; and

e. test data, including test databases.

2. The developer shall update the unit I&T regression test suite to reflect new and changed requirements, including interface requirements, and design.

3. The developer shall record the unit I&T regression test suite in the appropriate SDFs.

4. The developer shall execute the unit I&T regression test suite in accordance with Section 5.8.3 (performing) and Section 5.8.6 (retesting) after any changes, i.e., additions, deletions, or modifications, to:

a. any units that previously underwent unit I&T;

b. any reusable software included or integrated with the software units;

c. the target computer system(s); or

d. the software integration and qualification test environment.

5. The developer shall execute the unit I&T regression test suite after other software units have been added to the previously integrated and tested software units.

6. The developer shall execute the unit I&T regression test suite for developer-internal software integration testing.

   Note: For software developed in builds, this requirement will be performed on a build-by-build basis.

7. The developer shall automate the unit I&T regression test suite to the extent feasible.

   Note: See Section 5.8.4 for the requirements for analyzing and recording the unit I&T activities and results.

5.8.6 Revising and Retesting Unit Integration

1. Based on the results of unit I&T, including regression testing and retesting, the developer shall make all necessary revisions to the:

a. software;

b. instructions for preparing the test environment;

c. test cases for all verification methods used for unit I&T retesting;

d. test procedures;

e. test drivers or test scripts, or both;

f. test data, including test databases;

g. regression test suite; and

h. test environment.

2. The developer shall update the SDFs as needed to reflect the revisions for unit I&T.
3. The developer shall update other software products as needed to reflect the revisions for unit I&T.

4. The developer shall retest the integrated software units:
   a. in accordance with Section 5.8.3 (performing) and Section 5.8.4 (analyzing and recording), and
   b. in accordance with the applicable test cases and test procedures prepared according to Section 5.8.2 (preparing), Section 5.8.5 (regression), and Section 5.8.6 (retesting).

Note: See Section 5.8.4 for the requirements for analyzing and recording the unit I&T activities and results.

5.9 Software Item Qualification Testing

This section specifies the developer requirements for software item qualification testing (SIQT). Software item qualification testing is performed to verify that the software item requirements have been met. Software item qualification testing can also be performed to demonstrate to the acquirer or other stakeholders that software item requirements have been met. The stakeholders required to witness SIQT depend upon contract provisions. SIQT addresses the verification of software item requirements in Software Requirements Specifications (SRSs) and the software-related interface requirements in Interface Requirements Specifications (IRSs). This software item qualification testing contrasts with developer-internal software item integration testing, performed as the final stage of unit integration and testing (I&T).

SIQT is usually performed on a software item before delivering it to the acquirer or for integration with interfacing software items or integration at a higher level of integration and testing, e.g., software-hardware item I&T. The developer can also perform SIQT when software items are not being delivered to the acquirer or for integration outside the software item.

The term “software item qualification testing” refers to the activity of verifying that the software item requirements, including the software interface requirements have been met. Software item qualification testing is an iterative process accomplished for each software item and includes preparing for testing the software item, dry running the software item test cases, testing the software item, analyzing and recording the test results, fixing problems in the software or test preparation products (e.g., test data, test procedures), and retesting until the software item qualification test cases, including all nominal and off-nominal test cases, all execute successfully, i.e., with actual results matching expected results. Requirements are also included in this section for regression testing the software item if changes (e.g., to the software or the target computer system(s)) are made after the software item has successfully completed software item qualification testing in order to determine whether the changes adversely impact the previously verified software requirements or software interface requirements.

The activities in Section 5.9.5, Performing Software Item Qualification Testing, are sometimes called “formal execution” or “run for record.”

If a software item is developed in multiple builds, its software item qualification testing will not be completed until the final build for that software item or possibly until later builds involving items with which the software item is required to interface. Software item qualification testing for each build is interpreted to mean planning and performing tests of the current build of each software item to ensure that the software item requirements to be implemented in that build have been met.

The use of the words “test” or “testing” in this section is distinct from the verification method of Test. The activity of software item qualification testing can require the use of all verification methods, i.e., Inspection (I), Analysis (A), Demonstration (D), and Test (T).

See Appendix E, Section E.3.4 for the Software Build Test Readiness Review (SBTRR) requirements.
5.9.1 Independence in Software Item Qualification Testing

1. The individual(s) responsible for qualification testing of a given software item shall be different individual(s) than those who performed detailed design, implementation, unit testing, or unit integration and testing of that software item.

Note: This requirement does not preclude the individuals who developed one software item from assisting the qualification testers in qualification testing the software item.

5.9.2 Testing on the Target Computer System

See the definition of target computer system in Section 3.1.

1. The developer shall perform SIQT using the target computer system(s) in the operational configuration.

2. The developer shall conduct all SIQT under conditions representative of those that the software will encounter in the operational environment (e.g., operational data constants, operational input and output data rates, operational scenarios).

3. The developer shall perform SIQT using actual interfaces whenever possible.

4. If using actual interfaces is not possible for SIQT, then the developer shall use validated high-fidelity simulations of the interfaces.

Note: See Section 5.2.2 for validation requirements for simulations, simulators, and other software used for verification of requirements.

5. SIQT shall be performed with the entire software item under test installed in the target computer system(s).

Note: For software developed in multiple builds, this requirement might not be met until the final build.

6. The target computer system(s) used for SIQT shall be in the operational software configuration, including all other software executing on the target computer system(s) (e.g., operating system, COTS software and any other reusable software, and other software items) in addition to the entire software item under test.

7. The following shall be documented in the Software Test Plan (STP) for SIQT:
   a. the configuration of all software, the software item under test and all the other software included on the target computer system(s);
   b. the target computer system(s) and their configurations;
   c. the hardware in the software item qualification test environment and its configuration, including calibration dates, as applicable;
   d. the software item qualification test environment software and its configuration; and
   e. the conditions representing the operational environment (e.g., operational data constants, operational input and output data rates, operational scenarios).

Note: See Section 6.2 for the STP DID identifier.

5.9.3 Preparing for Software Item Qualification Testing

1. The developer shall prepare SIQT test cases to verify all software requirements (e.g., in SRSs), and software-related interface requirements (e.g., in IRSs and Interface Control Documents), whether they are implemented by reusable software or newly developed software.

2. The developer shall prepare to perform SIQT on:
   a. all newly developed software, and
   b. all modified and unmodified reusable software, including COTS software.

3. The developer shall develop the following test preparation products for conducting SIQT:
   a. instructions for preparing the test environment;
   b. test cases for all verification methods used for SIQT;
10. Test procedures;
11. test drivers or test scripts, or both; and
e. test data, including test databases.
Note: See Section 4.2.3.3 for traceability requirements for SIQT.
4. The SIQT preparation products shall be in accordance with the Software Test Plan (STP) (see Section 5.1.2).
5. The developer shall define SIQT equivalence classes and representative sets of nominal and off-nominal conditions, including valid and invalid values; values just inside the boundary of acceptable range values, at the boundary, and just outside the boundary; and extreme values.
6. Using these representative sets of nominal and off-nominal conditions, the developer shall define SIQT test cases that address, as a minimum, correct execution of:
a. all end-to-end functional capabilities through the software item under test;
b. scenarios containing multiple users or functions that must execute concurrently, using normal and heavy operational workloads;
c. concurrent access of the same data, i.e., reading, adding, updating, and deleting data by multiple users or functions in the software item under test;
d. all integrated error and exception handling across the software item under test;
e. all fault detection, isolation, recovery (e.g., failover), and data capture and reporting;
f. all startup, termination, and restart conditions, when applicable;
g. all stress testing, including worst-case scenarios (e.g., extreme workloads, high frequency of inputs and events, large number of users, simulated failed hardware, missing or malfunctioning interfaces, tight timelines); and
h. all endurance testing using normal and heavy operational workloads.
7. Using these representative sets of nominal and off-nominal conditions, the developer shall define SIQT test cases that address the software requirements, or portions thereof, allocated to the software item under test, including, as a minimum, all:
a. software requirements;
b. software interface requirements internal to the software item;
c. software interface requirements external to the software item;
d. performance requirements, including timing and accuracy requirements;
e. computer hardware resource utilization measurement requirements (e.g., CPU, memory, storage, bandwidth); and
f. software specialty engineering requirements (e.g., supportability, testability, dependability, reliability, maintainability, availability, safety, security, and human system integration, including human factors engineering, as applicable).
8. The SIQT cases to be executed shall include at least one test case for each SIQT equivalence class of nominal and off-nominal conditions.
9. The developer shall define SIQT cases to determine whether the software item under test is impacted by unrequired functionality in COTS software or any other reusable software.
10. The developer shall generate and prepare the software from the controlled configuration management system for the software item qualification test environment, including:
a. executable software;
b. procedures, if any;
c. data files, including databases; and
d. other software files needed to install, operate, and test the software on its target computer system(s).
11. The developer’s software quality assurance (SQA) personnel shall verify the configurations of the hardware, software, and software item qualification test environment, including:
a. verifying that the software was generated following the configuration management procedures,
b. verifying that the software checksums match the configuration management checksums,
c. verifying that the build identifier of the software under test is the same as the build identifier for the software generated from the controlled configuration management system, if a software checksum is not available, and
d. verifying that the software item qualification test environment configuration matches that specified in the test procedures documented in the STD DID.

12. The developer shall provide the acquirer a minimum of two weeks of advance notice of the exact time and location of:
   a. formal SIQT dry runs;
   b. the Software Build Test Readiness Review (SBTRR) for SIQT; (See Appendix E.3.4.)
   c. initial SIQT execution; and
   d. resumption of any SIQT (formal dry runs, initial, regression, or retesting) after a break of more than two weeks.

Note 1: The developer provided the planned review and testing schedule earlier.
Note 2: The developer and acquirer can mutually agree upon a different advance notification length of time for the SBTRR and SIQT.

13. The developer shall automate the SIQT test cases, test procedures, test drivers, test scripts, and test data to the extent feasible.

14. The developer shall record the results of preparing for SIQT in the Software Test Description (STD), including all applicable items in the STD DID, as defined in the SDP (see Section 5.1.1).

Note: See Section 6.2 for the STD DID identifier.

5.9.4 Dry Run of Software Item Qualification Testing

1. The developer shall dry run the SIQT of each software item in accordance with the SIQT test cases and test procedures prepared according to Section 5.9.3 to demonstrate that:
   a. the test cases and test procedures are complete and accurate, and
   b. the software is ready for witnessed testing.

2. The developer shall dry run all of the SIQT test cases successfully, i.e., with actual results matching expected results, using the version of the software item undergoing SIQT before performing:
   a. SIQT execution, according to Section 5.9.5;
   b. regression testing, according to Section 5.9.7; and
   c. any retesting, according to Section 5.9.8.

5.9.5 Performing Software Item Qualification Testing

Note: The activities in this subsection are sometimes called “formal execution” or “run for record.”

1. A Software Build Test Readiness Review (SBTRR) shall be held before performing the SIQT. Note: See Appendix E, Section E.3.4 for the SBTRR requirements.

2. The developer shall perform SIQT of each software item in accordance with:
   a. the SIQT plan prepared according to Section 5.1.2, and
   b. the test cases and test procedures prepared according to Section 5.9.3.

3. The developer’s software quality assurance (SQA) personnel shall verify that all prerequisite conditions are met for each test case.

4. The developer’s SQA personnel shall witness the SIQT.

5. The developer shall suspend the SIQT if a situation occurs during SIQT that causes any of the suspension criteria defined during the software test planning (see Section 5.1.2) to be triggered.
6. The developer shall terminate the SIQT if a situation occurs during SIQT that causes any of the termination criteria defined during the software test planning (see Section 5.1.2) to be triggered.

7. The developer shall terminate the SIQT execution if any test incident or test interruption occurs that requires any change of software, hardware, test environment, or configuration.

8. The developer shall terminate SIQT after a suspension or termination only after:
   a. SQA personnel have verified the configurations of the software, hardware, and software item qualification test environment, and
   b. the resumption criteria defined during the software test planning are met (see Section 5.1.2).

9. The developer shall record corrections to test procedure steps as redlines in the test procedure.

10. As qualification testing proceeds, the developer shall record the SIQT results as annotations to the test procedures, including:
    a. the results of each test procedure step,
    b. all differences found between documented test procedure steps and actual test procedure steps as executed, and
    c. all differences found between the expected results documented in the test procedure and the actual results as executed.

11. For all SIQT performed, including dry runs, initial SIQT execution, regression testing, and retesting, the developer shall record:
    a. a complete SIQT test log of SIQT as it proceeds;
    b. in the SIQT test log, as a minimum, the test log information specified in Appendix F, Section F.2 for SIQT;
    c. in the SIQT test log all test interruptions during SIQT;
    d. in the SIQT test log all test incidents found during SIQT, whether or not they are resolved during this activity;
    e. in the SIQT test log all instances, if any, of test incidents with the same symptoms, whether or not they are resolved during this activity; and
    f. in the SIQT test log references to all discrepancy and change reports, whether or not they are resolved during this activity.

12. The developer shall repeat the SIQT in accordance with Section 5.9.6 (analyzing) and Section 5.9.8 (retesting) until all test cases and test procedures have been performed successfully, i.e., with actual results matching expected results.

5.9.6 Analyzing and Recording Software Item Qualification Test Results

1. For all SIQT performed, including dry runs, initial SIQT execution, regression testing, and retesting, the developer shall:
   a. analyze the SIQT results;
   b. record for each requirement identifier whether it passed or failed;
   c. record in discrepancy and change reports all discrepancies found during SIQT execution or analysis, whether or not they are resolved during this activity;
   d. record in each discrepancy and change report, as a minimum, the information specified in Appendix C, Section C.2 for SIQT;
   e. record and maintain a cumulative record of the verification status, i.e., fully verified, partially verified, not verified, of each software requirement, including software interface requirements, for:
all SIQT, including initial test execution, regression testing, and retesting;
(2) all test cases for all verification methods used for SIQT, i.e., Inspection (I), Analysis (A), Demonstration (D), and Test (T); and
(3) all levels in the verification testing hierarchy (e.g., build, software item, subsystem, and system) (see Section 5.1.2);
f. record the SIQT results in the Software Test Report (STR), including all applicable items in the STR DID, as defined in the SDP (see Section 5.1.1);
Note: See Section 6.2 for the STR DID identifier;
g. record the SIQT analysis results in the Software Test Report (STR), including all applicable items in the STR DID, as defined in the SDP (see Section 5.1.1); and
h. record all additional SIQT analysis results (e.g., from dry runs and regression tests) in the appropriate SDFs.

5.9.7 Software Item Qualification Regression Testing
1. The developer shall define a SIQT regression test suite, including:
   a. instructions for preparing the software item qualification test environment;
   b. test cases for all verification methods used for regression testing the software item;
   c. test procedures;
   d. test drivers or test scripts, or both; and
   e. test data, including test databases.
2. The developer shall update the SIQT regression test suite to include new and changed requirements, including software interface requirements, architecture, and design.
3. The developer shall automate the SIQT regression test suite to the extent feasible.
4. The developer shall record the SIQT regression test suite in the appropriate SDFs.
5. The developer shall execute the SIQT regression test suite in accordance with Section 5.9.5 (performing) and Section 5.9.8 (retesting) after any changes, i.e., additions, deletions, or modifications, to:
   a. any software in the software item that has been previously qualification tested,
   b. any reusable software included or integrated with the software item,
   c. any other software, including test software and other software items executing on the target computer system(s) in addition to the software item under test,
   d. the target computer system(s) or their configuration(s), or
   e. the SIQT environment.
6. All of the SIQT test cases shall be dry run successfully, i.e., with actual results matching expected results, on the latest version of the software item undergoing test before performing:
   a. SIQT execution, according to Section 5.9.5;
   b. SIQT regression testing, according to Section 5.9.7; and
   c. any SIQT retesting, according to Section 5.9.8.
7. At the culmination of any build that is to be delivered for integration with interfacing software items or integration at a higher level of integration and testing, e.g., software-hardware item integration and testing, delivered to operations, or formally delivered to the acquirer, the developer shall execute the SIQT regression test suite to demonstrate that any changes, i.e., additions, deletions, or modifications, to the software in that build have not affected any requirements verified in previous builds.
Note: See Section 5.9.6 for the requirements for analyzing and recording the SIQT activities and results.
5.9.8 Revising and Retesting Software Items

1. Based on the results of SIQT, including dry runs, initial software qualification test execution, regression testing, and retesting, the developer shall make necessary revisions to the:
   a. software;
   b. instructions for preparing the test environment;
   c. test cases for all verification methods used for retesting the software item,
   d. test procedures;
   e. test drivers or test scripts, or both;
   f. test data, including test databases;
   g. regression test suite; and
   h. software item qualification test environment.

2. The developer shall update the test documentation as needed to reflect the revisions for SIQT, including:
   a. SDFs,
   b. STDs, and
   c. STRs,

3. The developer shall update other software products as needed to reflect the revisions for SIQT.

4. The developer shall retest the software item:
   a. in accordance with Section 5.9.5 (performing) and Section 5.9.6 (analyzing and recording), and
   b. in accordance with the applicable SIQT test cases and test procedures prepared according to Sections 5.9.3 (preparing), Section 5.9.7 (regression), and Section 5.9.8 (retesting).

Note: See Section 5.9.6 for the requirements for analyzing and recording the SIQT activities and results.

5.10 Software-Hardware Item Integration and Testing

This section specifies the developer requirements for software-hardware item integration and testing. 
Software-hardware item integration and testing is performed to integrate two or more software items and hardware items and to test the resulting integrated software and hardware items to ensure that they work together as intended. This process continues until all software items and hardware items in the system are integrated and tested.

The term “software-hardware item integration and testing” or “software-hardware item I&T” refers to the activity of verifying the correct functioning of an integrated collection of software items and hardware items or portions thereof.

Software-hardware item I&T is an iterative process accomplished for combinations of software items and hardware items, and includes preparing for integrating the software and hardware items, preparing for testing the integrated software and hardware items, integrating the items, testing the integrated items, analyzing and recording the test results, fixing problems in the software, hardware, or test preparation products (e.g., test data, test procedures), and retesting until the software-hardware item I&T test cases, including all nominal and off-nominal test cases, all execute successfully, i.e., with actual results matching expected results. Requirements are also included in this section for regression testing the integrated items if changes (e.g., to the software, target computer system(s), or other hardware) are made after the integrated items have successfully completed software-hardware item integration and testing in order to determine whether the changes adversely impact the functioning or performance of the integrated items.

As software-hardware item integration and testing becomes more complete, the software-hardware integration and test environment evolves to be more and more like the operational environment.
The use of the words “test” or “testing” in this section is distinct from the verification method of Test. The activity of software-hardware item integration and testing can require the use of all verification methods, i.e., Inspection (I), Analysis (A), Demonstration (D), and Test (T).

If a system or software item is developed in multiple builds, software-hardware item I&T will not be complete until the final build. Software-hardware item I&T in each build is interpreted to mean integrating the current build of each software item with the current build of other software items and hardware items and testing the results. See Section 5.1.2 for the software-hardware integration sequence.

The last stage of this software-hardware item I&T is developer-internal system integration testing. If the system is developed in multiple builds, then the developer-internal system integration testing will occur on a build-by-build basis.

This software-hardware item I&T activity addresses software item-to-software item, and software item-to-hardware item I&T. It does not address hardware item-to-hardware item I&T.

See Section 1.2.5.2 for interpretation of “participate.”

5.10.1 Testing on the Target Computer System

Note: The developer is strongly encouraged to meet the requirements in Section 5.10.1 as early as possible in software-hardware item integration and testing.

See the definition of target computer system in Section 3.1.

1. Software-hardware item I&T shall be performed on the target computer system(s) in the operational configuration(s).
2. Software-hardware item I&T shall be performed using the operational hardware with which the software must interface.
3. The software-hardware item I&T shall use actual interfaces whenever possible.
4. If using actual hardware interfaces is not possible for software-hardware item I&T, then high-fidelity simulations of the interfaces shall be used.
   Note: See Section 5.2.2 for validation requirements for simulations, simulators, and other software used for verification of requirements.
5. All software-hardware item I&T shall be conducted under conditions representative of those that the software will encounter in the operational environment (e.g., operational data constants, operational input and output data rates, operational scenarios).

5.10.2 Preparing for Software-Hardware Item Integration and Testing

1. The developer shall prepare to perform software-hardware item I&T on:
   a. all newly developed software; and
   b. all modified and unmodified reusable software, including COTS software.
2. The developer shall prepare for integrating the software items and hardware items in accordance with the integration sequence recorded in the Software Master Build Plan (SMBP) (see Section 5.1.2).
3. The developer shall participate in developing the following test preparation products for conducting software-hardware item integration and testing:
   a. instructions for preparing the test environment;
   b. test cases for all verification methods used for software-hardware item I&T;
   c. test procedures;
   d. test drivers or test scripts, or both; and
   e. test data, including test databases.
   Note: See Section 4.2.3.3 for traceability requirements for software-hardware item I&T.
4. The developer shall define and record software-hardware item integration and testing equivalence classes and representative sets of nominal and off-nominal conditions, including valid and invalid values; values just inside the boundary of acceptable range values, at the boundary, and just outside the boundary; and extreme values.

5. Using these representative sets of nominal and off-nominal conditions, the developer shall define and record software-hardware item integration test cases that address the software-hardware item design, including, as a minimum, correct execution of:
   a. all algorithms;
   b. all end-to-end functional capabilities through the software and hardware items under test;
   c. all interfaces among the software and hardware items under test;
   d. all software interfaces external to the software and hardware items under test;
   e. scenarios containing multiple users or functions that must execute concurrently using normal and heavy operational workloads;
   f. concurrent access of the same data, i.e., reading, adding, updating, and deleting data by multiple users or functions in the software and hardware items under test, using normal and heavy operational workloads;
   g. all integrated error and exception handling across the software and hardware items under test;
   h. all fault detection, isolation, recovery (e.g., failover), and data capture and reporting;
   i. all startup, termination, and restart conditions, when applicable;
   j. all relevant stress conditions, including worst-case scenarios (e.g., extreme workloads, high frequency of inputs and events, large number of users, simulated failed hardware, missing or malfunctioning interfaces, tight timelines); and
   k. endurance testing using normal and heavy operational workloads.

6. Using these representative sets of nominal and off-nominal conditions, the developer shall define software-hardware item integration test cases that address the requirements, or portions thereof, allocated to the software and hardware items under test, including, as a minimum, all:
   a. software requirements;
   b. interface requirements among the software and hardware items under test;
   c. software interface requirements external to the software and hardware items under test;
   d. performance requirements, including timing and accuracy requirements;
   e. computer hardware resource utilization measurement requirements (e.g., CPU, memory, storage, bandwidth); and
   f. software specialty engineering requirements (e.g., supportability, testability, dependability, reliability, maintainability, availability, safety, security, and human system integration, including human factors engineering, as applicable).

Note: Testing external interfaces early in the integration sequence is encouraged.

7. The software-hardware I&T cases shall include at least one software-hardware item I&T case for each software-hardware item I&T equivalence class of nominal and off-nominal conditions.

8. The developer shall define software-hardware I&T cases to determine whether the software items and hardware items under test are impacted by unrequired functionality in COTS software or any other reusable software.

9. The developer shall generate and prepare the software from the controlled configuration management system for the software-hardware I&T environment, including:
   a. executable software;
   b. procedures, if any;
   c. data files, including databases; and
   d. other software files needed to install, operate, and test the software on its target computer system(s).

10. The developer shall participate in automating the software-hardware item integration test cases, test procedures, test drivers, test scripts, and test data to the extent feasible.
11. The developer shall participate in recording software-hardware item I&T preparation results.

12. The developer shall record software-related software-hardware item I&T preparation results in appropriate SDFs.

5.10.3 Performing Software-Hardware Item Integration and Testing

1. The developer shall integrate the software items and hardware items in accordance with the software-hardware item integration sequence recorded in the Software Master Build Plan (SMBP) (see Section 5.1.2).

2. The developer shall participate in performing testing of the integrated software and hardware items in accordance with the test cases and test procedures prepared according to Section 5.10.2.

3. For all software-hardware item I&T performed, including initial testing, regression testing, and retesting, the developer shall participate in recording:
   a. a complete software-hardware item I&T test log of the software-hardware I&T as the integration and testing proceeds;
   b. in this test log, as a minimum, the test log information specified in Appendix F, Section F.2 for software-hardware item I&T;
   c. in this test log all test incidents found during software-hardware item I&T, whether or not they are resolved during this activity;
   d. in this test log all instances, if any, of test incidents with the same symptoms, whether or not they are resolved during this activity;
   e. in this test log references to all discrepancy and change reports, whether or not they are resolved during this activity.

4. The developer shall participate in repeating the software-hardware I&T testing in accordance with Section 5.10.4 (analyzing and recording) and Section 5.10.6 (retesting) until all test cases and test procedures have been performed successfully, i.e., with actual results matching expected results.

5.10.4 Analyzing and Recording Software-Hardware Item Integration and Test Results

1. For all software-hardware item I&T performed, including initial testing, regression testing, and retesting, the developer shall:
   a. participate in analyzing the software-hardware item I&T results;
   b. participate in recording in discrepancy and change reports all discrepancies found during software-hardware item I&T, whether or not they are resolved during this activity;
   c. participate in recording in each discrepancy and change report, as a minimum, the information specified in Appendix C, Section C.2 for software-hardware item I&T;
   d. participate in recording the software-hardware item I&T results;
   e. record the software-related software-hardware item I&T results in the appropriate SDFs;
   f. participate in recording the software-hardware item I&T analysis results; and
   g. record the software-related software-hardware item I&T analysis results in the appropriate SDFs.

5.10.5 Software-Hardware Item Integration Regression Testing

1. The developer shall participate in defining a software-hardware item I&T regression test suite, including:
   a. instructions for preparing the test environment;
   b. test cases for all verification methods used for regression testing of software-hardware I&T;
c. test procedures;
d. test drivers or test scripts, or both; and
e. test data, including test databases.

2. The developer shall participate in updating the software-hardware item I&T regression test suite to reflect new and changed requirements, including software interface requirements, and design.

3. The developer shall record the software-related software-hardware item I&T regression test suite in the appropriate SDFs.

4. The developer shall participate in executing the software-hardware item I&T regression test suite in accordance with Section 5.10.3 (performing) and Section 5.10.6 (retesting) after any changes, i.e., additions, deletions, and modifications, to:
   a. any previously tested software,
   b. any reusable software included or integrated with the software and hardware items,
   c. any other software, including test software and other software items, executing on the target computer system(s) in addition to the software item under test,
   d. the target computer system(s) or their configuration(s), or
   e. any previously tested operational hardware that interfaces with the software, or
   f. the software-hardware item I&T test environment.

5. The developer shall participate in executing the software-hardware item I&T regression test suite after any other software or hardware items have been added to the previously integrated and tested software and hardware items.

6. The software-hardware item I&T regression test suite shall be executed after the initial loading of the operational data, including, but not limited to:
   a. flight constants;
   b. ground constants;
   c. databases;
   d. stored procedures; and
   e. stored code.

7. The software-hardware item I&T regression test suite shall be executed after loading any changes to the operational data.

8. At the culmination of any build that is to be delivered for integration at a higher level or formally delivered to the acquirer, the developer shall participate in executing the software-hardware item I&T regression test suite to demonstrate that any changes, i.e., additions, deletions, and modifications, to the software or hardware in that build have not affected any requirements verified in previous builds.

9. The developer shall participate in automating the software-hardware item I&T regression test suite to the extent feasible.

Note: See Section 5.10.4 for the requirements for analyzing and recording the software-hardware item I&T activities and results.

5.10.6 Revising and Retesting Software-Hardware Item Integration

1. Based on the results of software-hardware item I&T, including initial testing, regression testing, and retesting, the developer shall make the necessary revisions to the software.

2. Based on the results of software-hardware item I&T, including initial testing, regression testing, and retesting, the developer shall participate in making necessary revisions to the:
   a. instructions for preparing the test environment;
   b. test cases for all verification methods used for software-hardware I&T retesting;
   c. test procedures;
   d. test drivers or test scripts, or both;
5.11 System Qualification Testing

This section specifies the software-related developer requirements for system qualification testing. System qualification testing is performed to verify that system requirements have been met. System qualification testing can also be performed to demonstrate to the acquirer or other stakeholders that system requirements have been met. System qualification testing in this section is interpreted to mean the verification of all levels of requirements higher in the specification tree than the software requirements. It covers requirements in the system specification, the subsystem specifications, if applicable, and all other levels of requirements between the system and subsystem specifications and the software requirements specifications in the specification tree (e.g., element specifications, segment specifications), including interface requirements at all of these levels. This system qualification testing contrasts with developer-internal system integration testing, performed as the final stage of software-hardware item integration and testing.

The term “system qualification testing” refers to the activity of verifying that the system requirements, including the system interface requirements, have been met. System qualification testing is an iterative process accomplished for the system and includes preparing for testing the system, dry running the system qualification test cases, testing the system, analyzing and recording the test results, fixing problems in the system or test preparation products (e.g., test data, test procedures), and retesting until the system qualification test cases, including all nominal and off-nominal test cases, all execute successfully, i.e., with actual results matching expected results. Requirements are also included in this section for regression testing the system if changes (e.g., to the software, target computer system(s), other hardware, or environment) are made after the system has successfully completed system qualification testing in order to determine whether the changes adversely impact the previously verified system requirements or system interface requirements.

If a system is developed in multiple builds, system qualification testing will not be completed until the final build. System qualification testing for each build is interpreted to mean planning and performing tests of the current build of the system to ensure that the system requirements to be implemented in that build have been met.

The use of the words “test” or “testing” in this section is distinct from the verification method of Test. The activity of system qualification testing can require the use of all verification methods, i.e., Inspection (I), Analysis (A), Demonstration (D), and Test (T).

See Section 1.2.5.2 for interpretation of “participate.”
5.11.1 Independence in System Qualification Testing

1. The individual(s) responsible for fulfilling the requirements in this section shall be different individual(s) than those who performed detailed design or implementation of the software in the system.

5.11.2 Testing on the Target Computer System(s)

See the definition of target computer system in Section 3.1.

1. System qualification testing shall be performed with the following in their operational configurations:
   a. software items that have successfully passed software item qualification test;
   b. target computer system(s); and
   c. operational hardware that interfaces with the software.

2. All software-related system qualification testing shall be conducted under conditions representative of those that the software will encounter in the operational environment (e.g., operational data constants, operational input and output data rates, operational scenarios).

3. The software-related system qualification testing shall use actual interfaces whenever possible.

4. If using actual interfaces is not possible for software-related system qualification testing, then validated high-fidelity simulations of the interfaces shall be used.

5. All software-related system qualification testing shall be performed with the entire software item installed in the target computer system(s) for all software items included in the system test.

6. The target computer system(s) shall be in the operational software configuration, including all software executing on the target computer system(s) (e.g., operating system(s), COTS software and any other reusable software, and all software items).

7. The following shall be documented in the system test plan for software-related system qualification testing:
   a. the configuration of all software, including test software, on the target computer system(s);
   b. the target computer system(s) and their configuration(s);
   c. the operational hardware that interfaces with the software;
   d. the conditions representing the operational environment (e.g., operational data constants, operational input and output data rates, operational scenarios).

5.11.3 Preparing for System Qualification Testing

1. The developer shall participate in developing the following test preparation products for conducting software-related system qualification testing:
   a. instructions for preparing the test environment;
   b. test cases for all verification methods used for system qualification testing;
   c. test procedures;
   d. test drivers or test scripts, or both; and
   e. test data, including test databases.

   Note: See Section 4.2.3.3 for traceability requirements for system qualification testing.

2. These software-related system qualification test preparation products shall be in accordance with the system test plan (see Section 5.1.3).

3. The developer shall generate and prepare the software from the controlled configuration management system for the system qualification test environment, including:
a. executable software;
b. procedures, if any;
c. data files, including databases; and
d. other software files needed to install, operate, and test the software on its target computer system(s).

4. The developer shall verify the configuration of the software, including:
   a. verifying that the software was generated following the configuration management procedures,
   b. verifying that the software checksums match the configuration management checksums, and
   c. verifying that the build identifier of the software under test is the same as the build identifier for the software generated from the controlled configuration management system if a software checksum is not available.

5. The developer shall participate in automating the software-related system qualification test cases, test procedures, test drivers, test scripts, and test data to the extent feasible.

6. The developer shall participate in recording software-related system qualification testing preparation results.

7. The developer shall record the software-related system qualification testing preparation results in the appropriate SDFs.

5.11.4 Dry Run of System Qualification Testing

1. The developer shall participate in dry running all of the software-related system qualification test cases successfully, i.e., with actual results matching expected results, using the versions of the software items undergoing system qualification testing before performing:
   a. system qualification test execution, according to Section 5.11.5,
   b. regression testing, according to Section 5.11.7, and
   c. any retesting, according to Section 5.11.8.

5.11.5 Performing System Qualification Testing

Note: The activities in this subsection are sometimes called “formal execution” or “run for record.”

1. The developer shall participate in performing the software-related system qualification testing in accordance with:
   a. the system qualification test plan prepared according to Section 5.1.3, and
   b. the system qualification test cases and test procedures prepared according to Section 5.11.3.

2. The developer shall participate in recording corrections to software-related test procedure steps as redlines in the test procedure.

3. As testing proceeds, the developer shall participate in recording the software-related system qualification testing results as annotations to the test procedures, including:
   a. the results of each test procedure step,
   b. all differences found between documented test procedure steps and actual test procedure steps as executed, and
   c. all differences found between the expected results documented in the test procedure and the actual results as executed.

4. For all software-related system qualification testing performed, including all dry runs, initial system qualification test execution, regression testing, and retesting, the developer shall participate in recording:
   a. a complete system qualification test log of all system qualification testing;
b. in this test log, as a minimum, the test log information specified in Appendix F, Section F.2 for system qualification testing;
c. in this test log all test interruptions during system qualification testing;
d. in this test log all test incidents found during system qualification testing, whether or not they are resolved during this activity;
e. in this test log all instances, if any, of test incidents with the same symptoms, whether or not they are resolved during this activity; and
f. in this test log references to all discrepancy and change reports, whether or not they are resolved.

5. The developer shall participate in repeating the software-related system qualification testing in accordance with Section 5.11.6 (analyzing and recording) and Section 5.11.8 (retesting) until all software-related test cases and test procedures have been performed successfully, i.e., with actual results matching expected results.

5.11.6 Analyzing and Recording System Qualification Test Results
1. For all software-related system qualification testing performed, including all dry runs, initial system qualification test execution, regression testing, and retesting, the developer shall:
   a. record the software-related results of system qualification testing in the appropriate SDFs;
   b. participate in analyzing the software-related system qualification testing results;
   c. participate in recording in discrepancy and change reports all discrepancies found during system qualification testing that are potentially software-related, whether or not they are resolved during this activity;
   d. participate in recording in discrepancy and change reports, as a minimum, the information specified in Appendix C, Section C.2 for system qualification testing;
   Note: For the discrepancies that are clearly hardware-only problems, some of the information specified in Appendix C, Section C.2 might not apply.
   e. participate in recording the software-related system qualification testing analysis results; and
   f. record the software-related system qualification testing analysis results in the appropriate SDFs.

5.11.7 System Qualification Regression Testing
1. The developer shall participate in defining a software-related system qualification regression test suite, including:
   a. instructions for preparing the test environment;
   b. test cases for all verification methods used for system qualification regression testing;
   c. test procedures;
   d. test drivers or test scripts, or both; and
   e. test data, including test databases.
2. The developer shall participate in updating the software-related system qualification regression test suite to reflect new and changed requirements, including interface requirements, and design.
3. The developer shall participate in automating the software-related system qualification regression test suite to the extent feasible.
4. The developer shall record the software-related system qualification regression test suite in the appropriate SDFs.
5. The developer shall participate in executing the software-related system qualification regression test suite in accordance with Section 5.11.5 (performing) and Section 5.11.8 (retesting) after any changes, i.e., additions, deletions, and modifications, to:
   a. any software that previously underwent system qualification testing;
b. any reusable software included or integrated with the system;
c. any other software, including test software, executing on the target computer system(s) in addition to the software items under test;
d. the target computer system(s) or their configuration(s);
e. any operational hardware that interfaces with the software; or
f. the configuration(s) of the hardware under test.

6. The test cases from the software-related system qualification regression test suite shall be executed after the initial loading of the operational data, including, but not limited to:
   a. flight constants;
   b. ground constants;
   c. databases;
   d. stored procedures; and
   e. stored code.

7. The test cases from the software-related system qualification regression test suite shall be executed after loading any changes to the operational data.

Note: See Section 5.11.6 for the requirements for analyzing and recording the system qualification testing activities and results.

5.11.8 Revising and Retesting the System

1. Based on the results of software-related system qualification testing, including dry runs, initial system qualification test execution, regression testing, and retesting, the developer shall make necessary revisions to the software.

2. Based on the results of software-related system qualification testing, including dry runs, initial qualification test execution, regression testing, and retesting, the developer shall participate in making necessary revisions to the:
   a. software-related test cases for all verification methods used for system qualification retesting,
   b. software-related test procedures;
   c. test drivers or test scripts, or both;
   d. test data, including test databases; and
   e. software-related regression test suite.

3. The developer shall update the SDFs to reflect the software-related system qualification testing revisions.

4. The developer shall update other software products to reflect the system qualification testing revisions.

5. The developer shall participate in software-related system qualification retesting:
   a. in accordance with Section 5.11.5 (performing) and Section 5.11.6 (analyzing and recording); and
   b. in accordance with the applicable test cases and procedures prepared according to Section 5.11.3 (preparing), Section 5.11.7 (regression), and Section 5.11.8 (retesting).

Note: See Section 5.11.6 for the requirements for analyzing and recording the system qualification testing activities and results.

5.12 Preparing for Software Transition to Operations

This section specifies the developer requirements for software use, including transition to operations.

Note 1: If software is developed in multiple builds, the developer’s build planning identifies for each build what software, if any, is to be fielded, i.e., distributed, to user sites and the extent of fielding (for example, full fielding or fielding to selected evaluators only). Preparing for software use in each build is interpreted to include those activities necessary to carry out the fielding plans for that build.
Note 2: This standard does not cover hardware installation.
Note 3: See Section 1.2.5.7 and 1.2.5.8 for definitions of user and user site, respectively.
Note 4: Different user sites may require different configuration parameters and different configurations of the software, some unique to the specific user site or category of user site.

5.12.1 Preparing the Executable Software
1. The developer shall generate and prepare the executable software for each user site.
2. Except for COTS executable software, the developer shall generate and prepare the executable software for the user site(s) from the configuration managed versions of source files for delivery for those user site(s).
   Note: Some of the COTS software might be generated and prepared using the built-in configuration, system administrator, user default, and other options, settings, and preferences, e.g., user privileges. Settings for different users might vary depending upon their roles.
3. The developer shall prepare any additional files, e.g., batch files, command files, configuration files, data files, databases, or other files, needed to install or operate the software on the target computer system(s) for each user site.
4. The preparation result shall include all applicable items in the executable software section of the Software Product Specification (SPS) DID, as defined in the SDP (see Section 5.1.1).
   Note: See Section 6.2 for the SPS DID identifier.

5.12.2 Preparing Version Descriptions for User Sites
1. The developer shall identify and record the exact version of software prepared for each user site.
2. The developer shall prepare and record the software installation instructions for each user site.
3. The result of preparing the version description information for the user site(s) shall include all applicable items in the Software Version Description (SVD) DID, as defined in the SDP (see Section 5.1.1).
   Note: See Section 6.2 for the SVD DID identifier.

5.12.3 Preparing User Manuals
This subsection specifies the developer requirements for preparing user manuals.

Note: Few, if any, systems need all of the manuals in this section. The intent is for the acquirer, with input from the users and developer, to determine which manuals are appropriate for a given system and to require the development of only those manuals. Commercial or other manuals that contain the required information can be substituted for the manuals specified in this standard. The manuals in this section are normally developed in parallel with the software development activities described in Sections 5.1 through 5.11.
1. Preliminary versions of the manuals in this section shall be available before the SIQT Software Build Test Readiness Review (SBTRR) for use in software item qualification testing.

5.12.3.1 Software User Manuals
1. The developer shall identify and record information needed by users of the software.
2. The information needed by users shall include all applicable items in the Software User Manual (SUM) DID, as defined in the SDP (see Section 5.1.1)
   Note: See Section 6.2 for the SUM DID identifier.

5.12.3.2 Computer Operation Manuals
This subsection only applies to computers without commercial operations manuals or to computers when their commercial operations manuals are inadequate. Examples of computer hardware for which this section applies are the processors in special test equipment or flight testbeds.
1. The developer shall identify and record the information needed to operate the computers on which the software will run.

2. The information needed to operate the computers shall include all applicable items in the Computer Operation Manual (COM) DID, as defined in the SDP (see Section 5.1.1).
   
   Note: See Section 6.2 for the COM DID identifier.

5.12.4 Installation at User Sites

Note: If software is developed in multiple builds, the developer’s build planning identifies for each build what software, if any, to install at each user site. Preparing for software transition for each build is interpreted to include those activities necessary to carry out the transition to operations plans for that build.

1. The developer shall install the executable software at the acquirer-designated user site(s) using the installation procedure in the SVD.
   
   Note: See Section 6.2 for the SVD DID identifier.

2. The developer shall perform the checkout process and procedures on the executable software and related files at each acquirer-designated user site.
   
   Note: Checkout can use subsets of the software item qualification regression test suite.

3. The developer shall provide training to the software users.

5.13 Preparing for Software Transition to Maintenance

This section specifies the developer requirements for preparing for transition of software to maintenance.

Note 1: If software is developed in multiple builds, the developer’s planning identifies for each build what software, if any, is to be transitioned to the maintenance organization(s). Preparing for software transition for each build is interpreted to include those activities necessary to carry out the transition plans for that build.

Note 2: Software maintenance might be performed by the same organization(s) that developed the software or by different organization(s) (e.g., an acquirer maintenance organization, another development contractor, another organization within the company that developed the software, or some combination of organizations).

Note 3: There may be one or more designated maintenance sites for the software. This section addresses transition to all designated maintenance sites.

5.13.1 Preparing the Executable Software

1. The developer shall generate and prepare the executable software to be transitioned to the maintenance site(s).

2. Except for COTS executable software, the developer shall generate and prepare the executable software from the configuration-managed versions of source files for delivery for the maintenance site(s).
   
   Note: Some of the COTS software might be generated and prepared using the built-in configuration, system administrator, user default, and other options, settings, and preferences, e.g., user privileges. Settings for different users might vary depending upon their roles.

3. The developer shall prepare any additional files, e.g., batch files, command files, configuration files, data files, databases, or other files, needed to maintain, regenerate, install, and operate the software on the maintenance computers and target computer system(s) for the maintenance site(s).

4. The result of preparing the executable software shall include all applicable items in the executable software section of the Software Product Specification (SPS) DID, as defined in the SDP (see Section 5.1.1).
   
   Note: See Section 6.2 for the SPS DID identifier.
5.13.2 Preparing Source Files

1. The developer shall prepare the source files to be transitioned to the maintenance site(s).
2. The developer shall prepare any additional files, e.g., batch files, command files, configuration files, data files, databases, or other files, needed to maintain, regenerate, install, and operate the software on the target computer system(s) for the maintenance site(s).
3. The result of preparing the source files shall include all applicable items in the source file section of the Software Product Specification (SPS) DID, as defined in the SDP (see Section 5.1.1).

   Note 1: See Section 6.2 for the SPS DID identifier.
   Note 2: See Section 4.2.3.4 for traceability requirements for the source files.

5.13.3 Preparing Version Descriptions for the Maintenance Site(s)

1. The developer shall identify and record the exact version of software prepared for the maintenance site(s).
2. The developer shall prepare and record the software installation instructions for the maintenance site(s).
3. The result of preparing the version description information for the maintenance site(s) shall include all applicable items in the SVD DID, as defined in the SDP (see Section 5.1.1).

   Note: See Section 6.2 for the SVD DID identifier.

5.13.4 Preparing the ‘As Built’ Software Architecture, Design, and Related Information

1. The developer shall update the overall software architecture in the Software Architecture Description (SAD) to match the “as built” software.

   Note: See Section 6.2 for the SAD DID identifier.
2. The developer shall update the software item architecture of each software item in the SAD to match the “as built” software.
3. The developer shall update the design descriptions of each software item in the SPS to match the “as built” software.
4. The developer shall define and record:
   a. the methods to be used to verify copies of the software,
   b. other information needed to maintain the software, and
   c. the measured computer hardware resource utilization for the software item.
5. The result of preparing the “as built” software item design and related information shall include all applicable items in the Software Product Specification (SPS) DID, as defined in the SDP (see Section 5.1.1):

   Note 1: See Section 6.2 for the SPS DID identifier.
   Note 2: Requirements for the executable software and source files sections of the SPS DID are addressed in Section 5.13.1 and Section 5.13.2.
   Note 3: See Section 4.2.3.4 for traceability requirements for the SPS.

5.13.5 Updating the System/Subsystem Design Description

1. The developer shall participate in updating the system and subsystem design description to match the “as built” system.
2. The result of updating the system/subsystem design description shall include all applicable items in the System/Subsystem Design Description (SSDD) DID, as defined in the SDP (see Section 5.1.1).

   Note 1: See Section 6.2 for the SSDD DID identifier.
   Note 2: See Section 4.2.3.1 for traceability requirements for the SSDD.
5.13.6 Updating the Software Requirements

1. The developer shall update the software requirements and software interface requirements for each software item to match the “as built” system.
2. The result of updating the software requirements shall include all applicable items in the Software Requirements Specification (SRS) DID as defined in the SDP (see Section 5.1.1).
3. The result of updating the software interface requirements shall include all applicable items in the Interface Requirements Specification (IRS) DID as defined in the SDP (see Section 5.1.1).

Note 1: See Section 6.2 for the SRS and IRS DID identifiers.
Note 2: See Section 4.2.3.1 for traceability requirements for software requirements and software interface requirements.

5.13.7 Updating the System Requirements

1. The developer shall participate in updating the system requirements to match the “as built” system.
2. The developer shall participate in updating the system interface requirements to match the “as built” system.
3. The result of updating the system requirements shall include all applicable items in the System/Subsystem Specification (SSS) DID, as defined in the SDP (see Section 5.1.1).
4. The result of updating the system interface requirements shall include all applicable items in the Interface Requirements Specification (IRS) DID, as defined in the SDP (see Section 5.1.1).

Note 1: See Section 6.2 for the SSS and IRS DID identifiers.
Note 2: An interface could be specified in system or subsystem specifications, as applicable, or could be specified in interface requirements specifications or interface control documents (ICDs).
Note 3: If a system consists of subsystems, the activity in Section 5.13.7 is intended to be performed also for subsystems at all levels.
Note 4: See Section 4.2.3.1 for traceability requirements for system requirements and interface requirements.

5.13.8 Preparing Maintenance Manuals

This section specifies the developer requirements for preparing maintenance manuals.

Note: Few, if any, systems will need the manuals discussed in this section. The intent is for the acquirer, maintainer, and developer to determine which manuals are appropriate for a given system and to require the development of only those manuals. Commercial or other manuals that contain the required information can be substituted for the manuals specified in this standard. The manuals in this section supplement the SSDD, the SAD, and the SPSs, which serve as the primary sources of information for software maintenance. The user manuals cited in Section 5.12.3 are also useful to maintenance personnel.

5.13.8.1 Computer Programming Manuals

This subsection applies only to newly developed computer hardware that does not have commercial programming manuals available or to computer hardware for which the commercial programming manuals are inadequate.

1. The developer shall identify and record information needed to program the computers on which the software was developed or on which it will run.
2. The information for programming the computers shall include all applicable items in the Computer Programming Manual (CPM) DID, as defined in the SDP (see Section 5.1.1).

Note: See Section 6.2 for the CPM DID identifier.

---

5 This subsection is based on information from Section 5.13.6 of (EIA/IEEE J-016).
6 This subsection is based on information from Section 5.13.7 of (EIA/IEEE J-016).
5.13.8.2 Firmware Support Manuals

1. The developer shall identify and record information needed to program and reprogram any firmware devices in which the software will be installed.

2. The information for programming and reprogramming the firmware devices shall include all applicable items in the Firmware Support Manual (FSM) DID, as defined in the SDP (see Section 5.1.1).

Note: See Section 6.2 for the FSM DID identifier.

5.13.9 Transition to the Designated Maintenance Site(s)

1. The developer shall install the software transitioning to maintenance in the designated software maintenance site(s).

2. The developer shall perform the checkout process and procedures on the software transitioning to maintenance at each designated software maintenance site.

Note: Checkout can use subsets of the software item qualification regression test suite and other regression test suites.

3. The developer shall demonstrate to the acquirer that the software transitioning to maintenance can be regenerated, i.e., compiled, linked, and loaded into an executable product, and maintained using the hardware and software tools at each designated maintenance site.

4. The developer shall provide training to the designated software maintenance organization(s) as specified in the contract and in the software transition plan (see Section 5.1.5).

5. The developer shall provide other assistance to the designated software maintenance organization(s) as specified in the contract and in the software transition plan (see Section 5.1.5).

5.14 Software Configuration Management

This section specifies the developer requirements for software configuration management.

Note: If a system or software item is developed in multiple builds, the products of each build might be refinements of, or additions to, products of previous builds. Software configuration management for each build takes place in the context of the products and configuration controls in place at the start of the build.

5.14.1 Configuration Identification

1. The developer shall identify the entities to be placed under configuration control.

2. The entities to be placed under configuration control shall include, as a minimum:
   a. all products to be developed or used under the contract;
   b. all products specified by this standard;
   c. all elements of the software engineering environment; and
   d. all elements of the software integration and qualification test environment.

3. The developer shall assign a project-unique identifier to each software item to be placed under configuration control.

4. The developer shall assign a project-unique identifier to each additional entity to be placed under configuration control.

5. The identification scheme shall be at the level at which entities will be controlled, for example, computer files, electronic media, documents, software units, hardware items, and software items.

6. The identification scheme shall include the version, revision, and release status of each entity.

7. For each identified entity (e.g., source code, document, executable code) the developer shall:
   a. label each entity unambiguously with product identification; and
b. include configuration identification within the entity if the product identification cannot be determined by physical examination.

8. Configuration identification shall occur prior to the implementation of change control.

### 5.14.2 Configuration Control

1. The developer shall establish and implement procedures designating:
   a. the levels of control each identified entity must pass through (for example, author control, project control, acquirer control);
   b. when each entity is to be placed under configuration control;
   c. the persons or groups with authority to:
      (1) authorize changes, and
      (2) make changes at each level (for example, the programmer, analyst, software lead, project manager, acquirer); and
   d. the steps to be followed to:
      (1) request entry of an entity into configuration control,
      (2) request authorization for changes,
      (3) process change requests,
      (4) assess impact of change requests,
      (5) track changes,
      (6) authorize distribution of changed products,
      (7) distribute changed products, and
      (8) maintain past versions.

Note 1: The levels of control to be implemented are dependent upon the entity to be placed under configuration control. Thus, an SRS generally passes through author-level control, software configuration management level control, and project-level control. Software code, on the other hand, generally has more levels of control, for example: individual author or engineer, software integration (e.g., team leader level), software build integration, software item qualification, subsystem, and system control. The SDP defines the levels of control to be used for each entity to be placed under configuration control, in addition to the roles, responsibilities and procedures for each level.

Note 2: The configuration control levels usually include one or more levels of configuration control boards (CCBs).

Note 3: See Section 5.17 for more information on discrepancy and change reports.

2. Changes that affect an entity already under acquirer control shall be proposed to the acquirer in accordance with contractually established forms and procedures, if any.

3. The developer shall establish and perform procedures for checking out and checking in entities from the configuration controlled library that ensure that:
   a. conflicting or simultaneous updates do not occur, and
   b. all changes are reviewed for impacts (e.g., correctness, safety, security) before being checked in.

4. The developer shall establish and perform archive and retrieval procedures to ensure that all past versions of all entities that have been placed under any level of configuration control above the individual author or engineer level can be retrieved only from the configuration controlled library.

Note: The goals of the archive and retrieval procedures are to ensure that:
   a. only authorized software is used, and
   b. the authorized software is archived and retrievable.

5. The developer shall control and maintain configuration entities throughout the system development lifecycle.

6. The developer shall establish and perform access control procedures that restrict individuals to the minimum set of configuration management functions and configuration entities needed to
perform their assigned duties (e.g., to access, establish, or change configuration controlled entities).

5.14.3 Configuration Status Accounting

1. The developer shall prepare and maintain records of the configuration status of all entities that have been placed under any level of configuration control above the individual author or engineer level.

2. The configuration status records shall include, as applicable for each entity:
   a. the current version, revision, and release of the entity,
   b. a record of changes to the entity since being placed under any level of configuration control above the individual author or engineer level, and
   c. the status of discrepancy and change reports (DCRs) affecting the entity.

3. The developer shall provide a build report for each build, including, as a minimum:
   a. the version of each configuration entity included,
   b. the DCRs closed with the build,
   c. the DCRs remaining open, and
   d. the changes, i.e., additions, deletions, and modifications, made.

   Note: The Software Version Description (SVD) DID has additional content that is useful for a build. See Section 6.2 for the SVD DID identifier.

4. The configuration status records shall be maintained throughout the system development lifecycle.

   Note: The degree of formality of the configuration status accounting can differ at different levels of configuration control.

5.14.4 Configuration Audits

1. The developer shall periodically conduct configuration audits at the times specified in the SDP of all entities that have been placed under any level of configuration control above the individual author or engineer level.

2. The configuration audits shall determine whether each entity incorporates all approved changes scheduled for inclusion at the time of the audit.

   Note: The approval of changes is usually performed by a software configuration control board.

3. The configuration audits for each entity shall determine whether the entity incorporates any unapproved changes.

4. Any configuration audit exception findings shall be documented in configuration management noncompliance reports that are then entered into the corrective action system. (See Section 5.17.)

   Note: The degree of formality and frequency of the configuration audits can differ at different levels of configuration control.

5.14.5 Packaging, Storage, Handling, and Delivery

1. The developer shall establish, record, and implement procedures for the packaging, storage, handling, and delivery of deliverable software products.

2. The developer shall maintain master copies of delivered software products throughout the system development lifecycle.

3. Throughout the system development lifecycle, the developer shall maintain master copies of the source files needed to generate the delivered executable software.

4. The developer shall establish procedures to maintain the integrity of the stored data as required by the contract, regardless of medium of storage throughout the system development lifecycle.

   Note: Techniques for maintaining the integrity of stored data include:

---

7 This subsection is based on information from Section 5.14.4 of (EIA/IEEE J-016).
a. selecting storage media that minimize regeneration errors or deterioration;
b. exercising and refreshing archived data at a frequency compatible with the storage life of the medium; and
c. storing duplicate copies in physically separate archives that minimize the risk of loss in the event of a disaster.

5. The **developer shall establish** the release authorization procedures, including approval authority, for releasing configuration items and other software **products** for:
   a. higher levels of integration,
   b. installation at user site(s), and
   c. delivery to the **acquirer**.

### 5.14.6 Baselines

1. The **developer shall establish** and **maintain** software **baselines** for internal use.  
2. The **developer shall establish** and **maintain** a software baseline for each delivery to higher levels of integration, e.g., subsystem, system.
3. The **developer shall establish** and **maintain** a software baseline for each delivery:
   a. to higher levels of integration,
   b. for installation in operations, and
   c. to the **acquirer**.
4. The **developer shall establish** and **maintain** an overall software architecture **baseline**.
5. The **developer shall establish** and **maintain** a software requirements **baseline** for each software **item**.
6. The **developer shall establish** and **maintain** a software architecture **baseline** for each software **item**.
7. The **developer shall establish** and **maintain** a software design **baseline** for each software **item**.
8. The **developer shall establish** and **maintain** a software item **baseline** for each software **item** for each **build**, including, as a minimum, a consistent set of the following configuration controlled items as **applicable** to the **build** and software **item**:
   a. software item requirements, including software interface requirements;
   b. software item architecture;
   c. software item design, interface design, and database design;
   d. software item unit test cases and results;
   e. software item source code files and other files necessary for generating the executable code and data files;
   f. software item executable files and data files;
   g. software item unit integration test cases, test procedures, and test results;
   h. software item qualification test cases, test procedures, and test results;
   i. software item version description, and installation and user documentation;
   j. software item maintenance documentation;
   k. software item “as built” documentation; and
   l. lists of requested and approved liens, waivers, and deviations for the software **item**;

   **Note:** For software developed in **builds**, this **requirement** will be performed on a **build-by-build** basis.

### 5.15 Software Peer Reviews and Product Evaluations

This section specifies the developer **requirements** for peer reviews and product evaluations.
5.15.1 Software Peer Reviews

This subsection specifies the developer requirements for performing peer reviews of software work products.

1. The developer shall perform peer reviews on all work products for the products listed in Appendix D Table D.3-1.

Note: Planning for peer reviews of software work products is part of software development planning (see Section 5.1.1) and is recorded in the SDP.

5.15.1.1 Plan for Software Peer Reviews

This subsection addresses planning for a group of peer reviews.

1. The developer shall plan for the peer reviews on the software work products as specified in the SDP.
2. The developer shall define and record what type of peer review to conduct on which types of software work products.
   Note: The type of software and work product influences the type of peer review required, e.g., the developer could require inspections for requirements peer reviews, inspections for design and code peer reviews of anomaly detection and correction units, and walkthroughs for nonmission-critical code.
3. The developer shall define and record which roles are required to participate in which types of peer reviews.
   Note: Required roles might include, e.g., systems engineers, user representatives, and other stakeholders for software requirements.
4. The developer shall define and record which roles must attend peer reviews for which types of software work products.
5. The developer shall define and record entry and exit checklists for each type of work product.
6. The developer shall define and record maximum sizes for each type of work product to enable the peer review coordinator to divide the work products into pieces for multiple peer review sessions.
7. The developer shall record the peer review planning for each type of work product in the SDP.
   Note: See Appendix D.3 for the requirements for evaluation criteria for work products and products.

5.15.1.2 Prepare for an Individual Peer Review

This subsection addresses preparation for an individual peer review.

1. For each peer review, the developer shall identify key reviewers who are required to participate in the peer review.
   Note: Key reviewers can be identified by name for specific work products.
2. For each peer review, the developer shall prepare for the type of peer review specified in the SDP for the software work product.
3. For each peer review, the developer shall demonstrate that the software work product satisfies the peer review entry criteria prior to distribution of peer review materials.
4. For each peer review, each reviewer shall prepare by reviewing the software work product prior to participating in the peer review.
   Note: See Appendix D.3 for the requirements for evaluation criteria for work products and products.

5.15.1.3 Conduct Peer Reviews

1. The developer shall perform peer reviews on the software work products as specified in the SDP, ensuring that all entry criteria are satisfied, including product size, adequate preparation time by each participant, and participation by all required participants.
2. For each peer review, the developer shall identify and record discrepancy and change reports, action items, and issues about the software work product. Note: See Appendix C.2 for discrepancy and change report definitions (C.2.1) and requirements (C.2.2).

3. For each peer review, the developer shall collect and record the peer review data, including the items listed in Section 5.15.1.4.

4. For each peer review, the developer shall demonstrate that the exit criteria for the peer review are satisfied.

5.15.1.4 Analyze and Report Peer Review Data
1. The developer shall analyze and report summary data about the peer reviews, including:
   a. preparation time;
   b. possible defects and issues found during preparation;
   c. length of peer review (all hours by all participants after preparation);
   d. defects found during the peer review;
   e. number of participants;
   f. roles of participants;
   g. conduct;
   h. results, including discrepancy and change reports, action items, and issues; and
   i. trends (e.g., the most frequent types of discrepancies).

5.15.2 Product Evaluations
This section specifies the developer requirements for performing product evaluations of the products generated in carrying out the requirements of this standard.
Note 1: If a system or software item is developed in multiple builds, the products of each build are evaluated in the context of the objectives established for that build. A product that meets those objectives can be considered satisfactory even though it is missing information designated for development in later builds.
Note 2: Planning for product evaluations is part of software development planning (see Section 5.1.1) and is recorded in the SDP.

5.15.2.1 In-Process and Final Product Evaluations
1. The developer shall perform in-process product evaluations of the products specified in Appendix D Table D.3-1.
2. The developer shall perform a final product evaluation of the completed products specified in Appendix D Table D.3-1.
   Note: If a system or software item is developed in builds, the final product evaluation is performed on a build-by-build basis as that product is completed in the context of the objectives for that build.
3. For each product specified in Appendix D Table D.3-1, the developer shall perform in-process evaluations and final evaluations using, as a minimum, the product evaluation criteria specified in Appendix D Table D.3-1 for that product.
   Note: The definitions for the evaluation criteria are specified in Appendix D.2.
4. The developer shall document all discrepancies found during the product evaluations in DCRs.
5. The developer shall manage the DCRs from the product evaluations using the corrective action system.

5.15.2.2 Product Evaluation Records
1. The developer shall prepare and maintain records of each product evaluation.
2. The product evaluation records shall contain, as a minimum, the following items:
   a. date of review;
b. product reviewed;
c. version of product reviewed;
d. size of product (e.g., pages, number of requirements);
e. evaluation criteria used;
f. participants by role and name;
g. preparation time by each participant;
h. length in time of review;
i. comment review matrix containing comments from all participants with dispositions for each;
j. discrepancy and change reports generated;
k. action items generated; and
l. issues identified.

3. The product evaluation records shall be maintained throughout the system development lifecycle.

4. Discrepancies in software products under any level of configuration control above the individual author or engineer level shall be managed as specified in Section 5.17, Corrective action.

5.15.2.3 Independence in Product Evaluation

1. The individuals responsible for evaluating a product shall be different individuals than those who developed the product.

   Note: This requirement does not preclude the individuals who developed the product from taking part in the evaluation (for example, as participants in a peer review of the product).

5.16 Software Quality Assurance

This section specifies the developer requirements for software quality assurance.

Note: If a system or software item is developed in multiple builds, the activities and software products of each build are evaluated in the context of the objectives established for that build. An activity or software product that meets those objectives can be considered satisfactory even though it is missing aspects designated for later builds. Planning for software quality assurance is included in software development planning (see Section 5.1.1) and is recorded in the SDP.

5.16.1 Software Quality Assurance Evaluations

1. The developer shall conduct ongoing evaluations at the times specified in the SDP of the software development processes, software products, work products, and software services to assess:
   a. adherence of the designated, performed processes to the applicable process descriptions, standards, and procedures in accordance with the:
      (1) contract, and
      (2) SDP;
   b. adherence of the designated software products to the applicable process descriptions, standards, and procedures in accordance with the:
      (1) contract, and
      (2) SDP;
   c. adherence of the designated work products to the applicable process descriptions, standards, and procedures in accordance with the:
      (1) contract, and
      (2) SDP;
   d. adherence of the designated software services to the applicable process descriptions, standards, and procedures in accordance with the:
(1) contract, and
(2) SDP;
e. that each software product required by this standard and each software product required by any other contract provisions:
(1) exists and has undergone software peer reviews;
(2) has undergone software product evaluations;
(3) has undergone testing, for those products where testing is applicable, and
(4) has undergone corrective action for all identified discrepancies.
f. continued consistency among all software-related plans, including as a minimum, the Software Development Plan (SDP), Software Test Plan (STP), and Software Master Build Plan (SMBP); and
g. continued consistency with system-level plans, including as a minimum, management plans and system test plans.
2. The developer shall provide feedback to affected groups on the:
a. evaluation results;
b. status of evaluation issues; and
c. status of DCRs.

5.16.2 Software Quality Assurance Records
1. The developer shall establish and maintain records of each software quality assurance activity.
2. The software quality assurance activity records shall be maintained throughout the system development lifecycle.

5.16.3 Independence in Software Quality Assurance
1. The individual(s) responsible for conducting software quality assurance evaluations of a given software product, work product, process, or service shall be different individual(s) than those:
a. who developed the software product, work product, process, or service, and
b. who are responsible for the software product, work product, process, or service.
2. The individual(s) responsible for assessing compliance with the contract and compliance with the processes shall have the:
a. resources,
b. responsibility,
c. authority, and
d. organizational autonomy necessary to:
   (1) permit objective software quality assurance evaluations, and
   (2) initiate and verify corrective actions.

5.16.4 Software Quality Assurance Noncompliance Issues
1. The developer shall communicate quality issues with the staff and managers.
2. The developer shall communicate noncompliance issues with the staff and managers.
3. The developer shall resolve quality issues with the staff and managers.
4. The developer shall resolve noncompliance issues with the staff and managers.
5. The developer shall use an established escalation mechanism for the appropriate level of management to resolve the issues.
6. The developer shall track noncompliance issues to resolution.
7. Noncompliance issues in software products or work products under any level of configuration control above the individual author or engineer level shall be managed as specified in Section 5.17.2, Corrective action system.
5.17 Corrective Action

This section specifies the developer requirements for performing corrective action.

5.17.1 Discrepancy and Change Reports

1. The developer shall establish and maintain discrepancy and change report (DCR) procedures that cover:
   a. writing and submitting DCRs into the corrective action system,
   b. prompt disposition of DCRs (e.g., assessment and assignment), and
   c. providing feedback to affected groups.

2. The developer shall prepare a DCR to describe:
   a. each discrepancy or test incident detected in software entities under any level of configuration control above the individual author or engineer level,
   b. each change requested in software entities under any level of configuration control above the individual author or engineer level,
   c. each discrepancy in activities or products required by the contract, and
   d. each discrepancy in activities or products specified in the SDP.

Note: See Appendix C.2 for discrepancy and change report definitions (C.2.1) and requirements (C.2.2).

3. The discrepancy and change reports shall serve as input to the corrective action system.

Note 1: The degree of formality of the discrepancy and change reports can differ at different levels of configuration control.

Note 2: Discrepancy and change reports are sometimes called problem reports, change requests, trouble reports, test incident reports, issue reports, and other terms.

5.17.2 Corrective Action System

This section specifies the requirements for the developer’s corrective action system.

1. The developer shall implement a corrective action system for managing:
   a. each discrepancy detected in software entities under any level of configuration control above the individual author or engineer level,
   b. each change requested in software entities under any level of configuration control above the individual author or engineer level,
   c. each discrepancy or issue in activities or products required by the contract, and
   d. each discrepancy or issue in activities or products specified in the SDP.

2. The developer shall demonstrate that:
   a. all discrepancy and change reports are promptly reported and entered into the corrective action system;
   b. the impacts are promptly evaluated;
      Note: Example impacts that can result include the effect on products (e.g., requirements, architecture, design, code, test cases), related products, cost, schedule, current and future releases, and operations.
   c. disposition (e.g., approved, rejected, deferred) is determined promptly;
   d. feedback and status are provided to affected groups;
   e. action is initiated on each discrepancy and change report;
   f. the change is tested or otherwise evaluated to determine:
      (1) whether the change is correctly implemented, and
      (2) whether the discrepancy has been fixed without introducing additional discrepancies;
   g. resolution is achieved;
   h. status is tracked to closure; and
i. records of the discrepancy and change reports are maintained throughout the system development lifecycle.

Note: See Appendix C.2 for discrepancy and change report definitions (C.2.1) and requirements (C.2.2).

3. The developer shall re-evaluate all discrepancy and change reports deferred from previous builds and software reviews to determine their current status.

4. The developer shall perform analysis to detect trends in the discrepancy and change reports.

5. The developer shall perform periodic evaluations of corrective actions to determine whether:
   a. discrepancies have been resolved,
   b. adverse trends have been reversed, and
   c. changes have been correctly implemented without introducing additional discrepancies.

6. The developer shall specify in the SDP the intervals and events for the periodic evaluations of corrective actions.

7. The developer shall report results of the periodic evaluations of corrective actions to management.

8. The developer shall review and follow up on corrective actions.

Note: The degree of formality of the discrepancy and change reports and the corrective action system can differ at different levels of corrective action.

5.18 Joint Technical and Management Reviews

This section specifies the developer requirements for joint (acquirer and developer) technical and management reviews.

Note: If a system or software item is developed in multiple builds, the types of joint reviews held and the criteria applied depend on the objectives of each build. Software products that meet those objectives can be considered satisfactory even though they are missing information designated for development in later builds.

5.18.1 Joint Technical Reviews

1. The developer shall plan and participate in joint technical reviews at locations and dates proposed by the developer and approved by the acquirer.

2. The joint technical review team shall include persons with technical knowledge of the domain.

3. The joint technical review team shall include persons with technical knowledge of the software products to be reviewed.

4. The joint technical reviews shall focus on in-process and final software products, rather than materials generated especially for the review.

Note: Additional requirements for joint technical reviews are specified in Appendix E, Joint Technical and Management Reviews. See E.2, E.3, and E.4 for objectives for Joint Technical Reviews.

5.18.2 Joint Management Reviews

1. The developer shall plan and participate in joint management reviews at locations and dates proposed by the developer and approved by the acquirer.

2. The joint management review team shall include persons with authority to make cost and schedule decisions.
3. The joint management reviews shall have the following objectives:
   a. Keep management informed about project status, directions being taken, technical agreements reached, and overall status of evolving software products.
   b. Review project status.
   c. Resolve issues that could not be resolved at joint technical reviews.
   d. Arrive at agreed-upon mitigation strategies for near- and long-term risks that could not be resolved at joint technical reviews.
   e. Identify and resolve management-level issues and risks not raised at joint technical reviews.
   f. Make recommendations for acquirer approvals needed for timely accomplishment of the project.

Note 1: Acquirer approval authority is required for some recommended actions.
Note 2: Additional requirements for joint management reviews are specified in Appendix E.5, Joint Management Reviews.

5.19 Software Risk Management
1. The developer shall perform software risk management throughout the system development lifecycle.
2. The developer shall:
   a. Identify, analyze, and prioritize the areas of the software development project that involve any potential software risks, whether technical, cost, or schedule risks;
   b. Develop strategies for managing those software risks;
   c. Record the software risks and strategies as specified in the SDP;
   d. Implement the strategies in accordance with the plan; and
   e. Track risks to closure.

5.20 Software Measurement
1. The developer shall plan, record, and perform software measurement throughout the system development lifecycle in accordance with ISO 15939, Systems and Software Engineering Measurement Process (ISO 15939).
2. The developer shall plan, record, and perform software measurement throughout the system development lifecycle following the guidance in (Abelson 2011).

5.20.1 Software Measurement Planning
1. The developer shall plan and record the project’s software measurement approach for collecting, analyzing, interpreting, applying, and reporting each software measurement.
2. The measurement system specified in the planning shall be aligned with the other contractual measurement systems following the guidance specified in the Measurement Systems Alignment section of (Abelson 2011).
   Note: These other contractual measurement systems include, e.g., Work Breakdown Structure (WBS), Integrated Master Schedule (IMS), risk management.
3. The Software Measurement Plan (SMP) for the software measurement activities required by this standard shall use the items in the Software Measurement Plan (SMP) template in Appendix H.4 Software Measurement Plan (SMP) Template as guidance for the SMP.
   Note: See Section 6.2 for the SMP template identifier.

Note: The exact contractual requirements for deliverable products are in the contract. Those contractual requirements state where the software measurement planning information is to be recorded. If the software measurement plan is not a deliverable, then the software measurement planning information may be included in Section 5.20 of the SDP or in a separate software measurement plan.
5.20.2 Software Measurement Reporting

1. The developer shall perform the measurement process by collecting, analyzing, interpreting, applying, and reporting the required base measures, indicators, and derived measures specified in the Software Measurement Plan (SMP).
2. The developer shall perform the measurement process at the times specified in the SMP.
3. Measurement reporting shall be performed for each software item.
4. Measurement reporting shall be performed for each build.
5. The Software Measurement Report (SMR) shall use the items in the Software Measurement Report (SMR) template in Appendix H.5 Software Measurement Report (SMR) Template as guidance for the SMR.

Note: See Section 6.2 for the SMR template identifier.

5.20.3 Software Measurement Working Group (SMWG)

1. A joint acquirer and developer software measurement working group (SMWG) shall be established to accommodate changing measurement needs throughout the system development lifecycle.

5.21 Security and Privacy

1. The developer shall meet the security requirements and the privacy requirements specified in the contract.

Note: These requirements can affect any combination of the software development methods, the software product standards, and the resulting software products.

5.22 Software Team Member Management

1. The developer shall include all software-related contract provisions from the acquirer’s contract with the prime contractor into the agreements between all software team members performing software-related work on the contract.
2. Each software team member shall enforce the compliance of all subordinate software team members with the software-related contract provisions that have been placed in their respective agreements.

5.23 Interface with Software IV&V Agents

1. The developer shall interface with the software Independent Verification and Validation (IV&V) agent(s) as specified in the contract.

5.24 Coordination with Associate Developers

1. The developer shall coordinate with associate developers, working groups, and interface groups as specified in the contract.

5.25 Improvement of Project Processes

1. The developer shall define the method to be used to assess the processes in use on the project to determine their suitability, efficiency, and effectiveness.
2. The developer shall document in the SDP:
   a. the assessment method, and
   b. the periodic assessment schedule.
3. The developer shall periodically assess the processes used on the project to determine their suitability, efficiency, and effectiveness, as specified in the SDP.
4. Based on the assessment results, the developer shall perform the following process improvement planning:
   a. identify any necessary and beneficial improvements to the processes,
   b. plan for corrections to and improvements to the project’s processes, and
   c. schedule the improvements and subsequent assessments.
5. The developer shall document the results of the process improvement planning in the Process Improvement Plan (PIP).
6. The resulting process improvement planning shall include all applicable items in the PIP template in Appendix H.6 Process Improvement Plan (PIP) template, as defined in the SDP (see Section 5.1.1).
   Note: See Section 6.2 for the PIP template identifier.
7. The developer shall execute process improvement in accordance with the PIP.
8. The developer shall update the project’s processes per the improvements identified in the PIP.
9. The developer shall update the SDP as needed to include or reference the updated project processes.
10. The developer shall include assessment of the updated project processes in subsequent periodic assessments to determine whether the developer has achieved the expected beneficial improvements.
11. The developer shall update the PIP for each periodic assessment.
6. Notes

6.1 Intended Use
This section contains information of a general or explanatory nature that might be helpful, but is not mandatory.

6.2 Data Item Descriptions (DIDs)
The following DIDs are listed, as applicable, on the Contract Data Requirements List (CDRL) in order to have the products delivered under the contract.

<table>
<thead>
<tr>
<th>DID Title</th>
<th>DID or Template Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Development Plan (SDP)</td>
<td>DI-IPSC-81427A with Appendix H.1 content</td>
</tr>
<tr>
<td>Software Test Plan (STP)</td>
<td>DI-IPSC-81438A</td>
</tr>
<tr>
<td>Software Installation Plan (SIP)</td>
<td>DI-IPSC-81428A</td>
</tr>
<tr>
<td>Software Transition Plan (StrP)</td>
<td>DI-IPSC-81429A</td>
</tr>
<tr>
<td>Operational Concept Description (OCD)</td>
<td>DI-IPSC-81430A</td>
</tr>
<tr>
<td>System/Subsystem Specification (SSS)</td>
<td>DI-IPSC-81431A</td>
</tr>
<tr>
<td>Interface Requirements Specification (IRS)</td>
<td>DI-IPSC-81434A</td>
</tr>
<tr>
<td>System/Subsystem Design Description (SSDD)</td>
<td>DI-IPSC-81432A</td>
</tr>
<tr>
<td>Interface Design Description (IDD)</td>
<td>DI-IPSC-81436A</td>
</tr>
<tr>
<td>Software Requirements Specification (SRS)</td>
<td>DI-IPSC-81433A</td>
</tr>
<tr>
<td>Software Design Description (SDD)</td>
<td>DI-IPSC-81435A</td>
</tr>
<tr>
<td>Database Design Description (DBDD)</td>
<td>DI-IPSC-81437A</td>
</tr>
<tr>
<td>Software Test Description (STD)</td>
<td>DI-IPSC-81439A</td>
</tr>
<tr>
<td>Software Test Report (STR)</td>
<td>DI-IPSC-81440A</td>
</tr>
<tr>
<td>Software Product Specification (SPS)</td>
<td>DI-IPSC-81441A</td>
</tr>
<tr>
<td>Software Version Description (SVD)</td>
<td>DI-IPSC-81442A</td>
</tr>
<tr>
<td>Software User Manual (SUM)</td>
<td>DI-IPSC-81443A</td>
</tr>
<tr>
<td>Computer Operation Manual (COM)</td>
<td>DI-IPSC-81446A</td>
</tr>
<tr>
<td>Computer Programming Manual (CPM)</td>
<td>DI-IPSC-81447A</td>
</tr>
<tr>
<td>Firmware Support Manual (FSM)</td>
<td>DI-IPSC-81448A</td>
</tr>
<tr>
<td>Software Architecture Description (SAD)</td>
<td>DI-MISC-80508B with Appendix H.2 content</td>
</tr>
<tr>
<td>Software Master Build Plan (SMBP)</td>
<td>DI-MGMT-80004A with Appendix H.3 content</td>
</tr>
<tr>
<td>Software Measurement Plan (SMP)</td>
<td>DI-MGMT-80004A with Appendix H.4 content</td>
</tr>
<tr>
<td>Software Measurement Report (SMR)</td>
<td>DI-MISC-80508B with Appendix H.5 content</td>
</tr>
<tr>
<td>Process Improvement Plan (PIP)</td>
<td>DI-MGMT-80004A with Appendix H.6 content</td>
</tr>
</tbody>
</table>

Note 1: Depending on CDRL item provisions, requirements concerning system interfaces can be included in the SSS or in the Interface Requirements Specifications (IRFs).
Note 2: To contractually require only the executable software (delaying delivery of source files and associated support information to a later build), the acquirer can use the SPS DID, tailoring out all but the executable software section of that DID.
6.3 Deliverable Versus Nondeliverable Software Products

6.3.1 Philosophy of the Standard
This standard has been worded to differentiate between the planning and engineering activities that make up a software development project and the generation of deliverables. A key objective of this wording is to eliminate the notion that the acquirer needs to contractually require a given deliverable in order to have planning or engineering work take place. Under this standard, the planning and engineering work takes place regardless of which deliverables are contractually required, unless a given activity is tailored out of the standard. In addition, joint technical reviews have been included to review the results of that work in its natural form, without the generation of deliverables.

6.3.2 Contracting for Deliverables
Deliverables are contractually required only when there is a genuine need to have planning or engineering information transformed into a deliverable, recognizing that this transformation requires time and effort. The Description, Purpose, Use, or Relationship paragraph in a DID provides information that is helpful in deciding whether the corresponding deliverable is to be contractually required. The acquirer needs to consider future needs as well as the needs of the current contract when deciding which software products to make deliverable (e.g., future contracting for software maintenance or reprocurement). The acquirer also needs to consider future needs when establishing the data rights needed by the acquirer for the deliverable products. The data rights that the acquirer requires need to be specified in the contract.

6.3.3 Scheduling Deliverables
This standard has been structured to support a variety of development strategies and to provide the developer with flexibility in laying out a software development process that will best suit the work to be done. All of this flexibility can be canceled by rigid scheduling of deliverables in the contract. If the contract lays out a strict “waterfall” sequence of deliverables, little room is left to propose innovative development processes. If the contract forces all software items into lockstep with each other, little room is left to develop the software items in an optimum order. To the maximum extent possible, the acquirer needs to avoid such predetermination, allowing the developer to incrementally deliver software products, stagger the development of software items, and use other variations to optimize the software development effort. The developer’s detailed software schedule that meets the constraints in the contract is documented in the SDP.

6.3.4 Format of Deliverables
Traditional deliverables (such as CDRL items based on DIDs) take the form of paper documents exactly following required formats and structure. While this form works well for some deliverables, it is not the only form, and alternatives need to be considered. One variation from paper documents is word processing files containing those documents. This format saves paper but still requires the developer to format and structure the information as required. Another variation is specifying that a paper or word processor document is to include all required contents but is allowed to be in the developer’s format.

Yet another variation is allowing deliverables to take forms that are not traditional documents at all, such as data in computer-aided engineering (CAE) tools. These variations in required format can be specified in the contract, minimizing the time spent transforming actual work products into paper or electronic deliverables.

6.5 Tailoring Guidance
This standard and its DIDs are applied at the discretion of the acquirer. The acquirer is expected to tailor the standard and DIDs to the specific requirements of a particular project, acquirer’s program phase, and contractual structure. Tailoring for the standard takes the form of deleting activities, modifying activities to more explicitly reflect the application to a particular effort, or adding activities.
to satisfy the *acquirer’s requirements*. This tailoring is specified in the *Statement of Work (SOW)* or Compliance Documents section of the *contract*. Tailoring for the *DIDs* consists of deleting *requirements* for unneeded information and making other changes, such as explanations of the meaning of the *DID* language, that do not increase the required workload. *DID* tailoring for an individual deliverable *product* is specified in the *contractual* form to the *CDRL* item.

In order for the *acquirer* to have approval authority for a particular *CDRL* item, the *acquirer* places the appropriate approval code in the *contractual* form for that *CDRL* item. (For DOD contracts, the DD 1423 form is used to place a *CDRL* item on *contract*.)

### 6.6 Related Standardization Documents

Other standards can be imposed or quoted in the *SOW* or Compliance Documents section of the *contract* to supplement the *requirements* in this standard. The *acquirer* needs to use caution to ensure that supplemental standards are appropriate to the project and that any conflicts among them or with this standard are identified and resolved.
Appendix A. List of Acronyms and Abbreviations

A.1 Scope
This appendix provides a list of acronyms and abbreviations used in this standard. This appendix is not a mandatory part of the standard. The information provided is intended for guidance only.

A.1.1 Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym or Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>§</td>
<td>section or paragraph</td>
</tr>
<tr>
<td>A</td>
<td>Analysis</td>
</tr>
<tr>
<td>ADS</td>
<td>appraisal disclosure statement</td>
</tr>
<tr>
<td>API</td>
<td>application programming interface</td>
</tr>
<tr>
<td>ARC</td>
<td>appraisal requirements for CMMI®</td>
</tr>
<tr>
<td>ASIC</td>
<td>application specific integrated circuit</td>
</tr>
<tr>
<td>CAE</td>
<td>computer-aided engineering</td>
</tr>
<tr>
<td>CAR</td>
<td>causal analysis and resolution</td>
</tr>
<tr>
<td>CASE</td>
<td>computer-aided software engineering</td>
</tr>
<tr>
<td>CDR</td>
<td>critical design review</td>
</tr>
<tr>
<td>CDRL</td>
<td>Contract Data Requirements List</td>
</tr>
<tr>
<td>CIO</td>
<td>chief information officer</td>
</tr>
<tr>
<td>CMMI®</td>
<td>Capability Maturity Model® Integration</td>
</tr>
<tr>
<td>CMMI®-DEV</td>
<td>Capability Maturity Model® Integration for Development</td>
</tr>
<tr>
<td>CMU</td>
<td>Carnegie Mellon University</td>
</tr>
<tr>
<td>CNSS</td>
<td>Committee on National Security Systems</td>
</tr>
<tr>
<td>CNSSI</td>
<td>CNSS Instruction</td>
</tr>
<tr>
<td>COM</td>
<td>Computer Operation Manual</td>
</tr>
<tr>
<td>COTS</td>
<td>commercial off-the-shelf</td>
</tr>
<tr>
<td>CSCI</td>
<td>computer software configuration item</td>
</tr>
<tr>
<td>CSU</td>
<td>computer software unit</td>
</tr>
<tr>
<td>CPM</td>
<td>Computer Programming Manual</td>
</tr>
<tr>
<td>CPU</td>
<td>computer processing unit</td>
</tr>
<tr>
<td>D</td>
<td>Demonstration</td>
</tr>
<tr>
<td>DAU</td>
<td>Defense Acquisition University</td>
</tr>
<tr>
<td>DBDD</td>
<td>Database Design Description</td>
</tr>
<tr>
<td>DCR</td>
<td>discrepancy and change report</td>
</tr>
<tr>
<td>DEV</td>
<td>Development</td>
</tr>
<tr>
<td>DID</td>
<td>Data Item Description</td>
</tr>
<tr>
<td>DOD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>EIA</td>
<td>Electrical Industries Association</td>
</tr>
<tr>
<td>FAR</td>
<td>Federal Acquisition Regulations</td>
</tr>
<tr>
<td>FPGA</td>
<td>field-programmable gate array</td>
</tr>
<tr>
<td>FMEA</td>
<td>failure modes and effects analysis</td>
</tr>
<tr>
<td>FSM</td>
<td>Firmware Support Manual</td>
</tr>
<tr>
<td>GUI</td>
<td>graphical user interface</td>
</tr>
<tr>
<td>I&amp;T</td>
<td>integration and testing</td>
</tr>
<tr>
<td>I/O</td>
<td>input/output</td>
</tr>
<tr>
<td>I</td>
<td>Inspection</td>
</tr>
<tr>
<td>IA</td>
<td>information assurance</td>
</tr>
<tr>
<td>ICD</td>
<td>Interface Control Document</td>
</tr>
</tbody>
</table>
ID    identification
IDD   Interface Design Description
IEC   International Electrotechnical Commission
IEEE  Institute of Electrical and Electronic Engineers
IMP   integrated master plan
IMS   integrated master schedule
IRS   Interface Requirements Specification
ISO   International Organization for Standardization
IV&V  Independent Verification and Validation
KPP   Key Performance Parameter
MIL   military
MOSA  modular open software approach
MOU   memorandum of understanding
OCD   Operational Concept Description
OPM   organizational performance management
OPP   organizational process performance
OSS   open source software
PDR   preliminary design review
PIP   Process Improvement Plan
PLD   programmable logic device
QPM   quantitative project management
RMA   reliability, maintainability, and availability
SAD   Software Architecture Description
SAR   software requirements and architecture review
SBDR  software build design review
SBER  software build exit review
SBPR  software build planning review
SBRAR software build requirements and architecture review
SBTRR software build test readiness review
SCAMPI\textsuperscript{(SM)} Standard CMMI\textsuperscript{®} Appraisal Method for Process Improvement
SDD   Software Design Description
SDF   software development file
SDL   software development library
SDP   Software Development Plan
SDR   system design review
SDSMCS Software Development Standard for Mission Critical Systems
SEI   Software Engineering Institute
SFR   system functional review
SI    software item
SIP   Software Installation Plan
SIQT  software item qualification testing
SMBP  Software Master Build Plan
SMP   Software Measurement Plan
SMR   Software Measurement Report
SMWG  software measurement working group
SOW   statement of work
SPD   Software Product Development (TAI course)
SPS   Software Product Specification
SQA   software quality assurance
SRR   system requirements review
SRS   Software Requirements Specification
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSDD</td>
<td>System/Subsystem Design Description</td>
</tr>
<tr>
<td>SSS</td>
<td>System/Segment Specification or System/Subsystem Specification</td>
</tr>
<tr>
<td>STD</td>
<td>software test description</td>
</tr>
<tr>
<td>Std</td>
<td>standard</td>
</tr>
<tr>
<td>STD</td>
<td>standard</td>
</tr>
<tr>
<td>STP</td>
<td>Software Test Plan</td>
</tr>
<tr>
<td>STR</td>
<td>Software Test Report</td>
</tr>
<tr>
<td>StrP</td>
<td>Software Transition Plan</td>
</tr>
<tr>
<td>SUM</td>
<td>Software User Manual</td>
</tr>
<tr>
<td>SVD</td>
<td>Software Version Description</td>
</tr>
<tr>
<td>T</td>
<td>Test</td>
</tr>
<tr>
<td>TAI</td>
<td>The Aerospace Institute</td>
</tr>
<tr>
<td>TBD</td>
<td>to be determined</td>
</tr>
<tr>
<td>TBR</td>
<td>to be resolved</td>
</tr>
<tr>
<td>TBS</td>
<td>to be supplied</td>
</tr>
<tr>
<td>TBX</td>
<td>TBD, TBR, or TBS</td>
</tr>
<tr>
<td>TCS</td>
<td>target computer system</td>
</tr>
<tr>
<td>TOR</td>
<td>technical operating report</td>
</tr>
<tr>
<td>TPM</td>
<td>technical performance measure</td>
</tr>
<tr>
<td>TRR</td>
<td>test readiness review</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>UOM</td>
<td>unit of measure</td>
</tr>
<tr>
<td>WBS</td>
<td>work breakdown structure</td>
</tr>
</tbody>
</table>
Appendix B. Interpreting This Standard for Incorporation of COTS and Other Reusable Software Products

B.1 Scope
This appendix is a MANDATORY part of this standard. It provides evaluation criteria for reusable software products.

See Section 3.1 for the definitions of terms in italics, such as supplier and reusable software, which includes COTS software.

B.2 Evaluating Reusable Software Products
See Section 3.1 for the definition of the term “supplier.” Criteria for evaluating reusable software products shall include the following, as a minimum:

1. Ability to provide required capabilities and meet required constraints:
   a. Ability to satisfy requirements;
   b. Ability to achieve necessary performance, especially with realistic operational workloads;
   c. Appropriateness of algorithms in the reusable software product for use in the new system; and
   d. As evidenced by characterization testing, stress testing, and prototyping within the system context to determine capabilities and performance.

2. Ability to provide required protection, i.e., safety, security, and privacy protection:
   a. Inherently in the reusable software product,
   b. Around the reusable software product by system design and implementation, or
   c. The current product version is on an approved list of validated products for the domain and environment.

3. Reliability and maturity:
   a. As evidenced by prototype evaluation within the system context, and
   b. As evidenced by established track record (e.g., discrepancy and change report history).

4. Testability:
   a. As evidenced by the ability to identify and isolate faults; and
   b. Feasibility of testing or otherwise assessing complete conformance to requirements for behavior under nominal and off-nominal conditions, failure behavior, and recovery behavior.

5. Operability:
   a. Suitability of the reusable software product’s implied operations concept to the operations concept of the new system; and
   b. Lack of functionality that would inhibit operations, such as a periodic need to enter in a license code or the presence of a physical key or similar device to enforce licensing conditions.

6. Viability of reusable software product supplier:
   a. Compatibility of supplier’s future direction for the reusable software with project needs, including both software and platform emphasis;
   b. Supplier long-term commitment to the reusable software product;
   c. Supplier long-term business prospects;
   d. Type of supplier support available; and
   e. Quality of supplier support available.

7. Suitability for incorporation into the new system architecture:
   a. Compatible software architecture and design features,
   b. Absence of obsolete technologies,
c. Lack of or minimal need for re-engineering or additional code development (e.g., wraps, “glue” code),

d. Compatibility among the set of reusable software products, and
e. As evidenced by prototyping within the system context (e.g., to determine compatibility, wraps, “glue” code).

8. Ability to remove or disable features or capabilities not required in the new system:
   a. Impact if those features cannot be removed or disabled, or are not removed or disabled, and
   b. As evidenced by prototyping within the system context.

9. Interoperability with other system elements and external systems:
   a. Compatibility with system interfaces,
   b. Adherence to standards (e.g., open systems interface standards), and
   c. Ability to interface with legacy systems.

10. Availability of personnel knowledgeable about the reusable software product:
    a. Training required,
    b. Hiring required, and
    c. Supplier or third-party support required.

11. Availability and quality of documentation and source files:
    a. completeness, and
    b. Accuracy.

12. Acceptability of reusable software product licensing and data rights:
    a. Restrictions on copying or distributing the reusable software product or documentation;
    b. License or other fees applicable to each copy;
    c. Acquirer’s usage and ownership rights, especially to the source code:
        (1) Ability to use the reusable software product as needed by the acquirer, operator, user, and maintainer; and
        (2) Ability to place source code in escrow against the possibility of its supplier going out of business;
    d. Warranties available; and
    e. Absence of unacceptable restrictions in standard license (e.g., export restrictions, expiring keys).

13. Supportability:
    a. Suitability of the reusable software product’s support paradigm (e.g., distribution, installation) to the support concept of the new system, especially for mobile or remote user sites.

14. Ability to make changes, including:
    a. Likelihood the reusable software product will need to be changed;
    b. Feasibility and difficulty of accomplishing that change when changes are to be made by the project reusing the software product:
        (1) Quality of design, code, and documentation; and
        (2) Need for re-engineering or restructuring, or both; and
    c. Feasibility and difficulty of accomplishing change when changes must be made by the supplier or product developer (e.g., for COTS software or proprietary software):
        (1) Priority of changes required by this project versus other changes being made,
        (2) Likelihood that the changed version will continue to be maintained by its supplier or developer,
        (3) Likelihood of being able to modify future versions to include changes,
        (4) Impact on lifecycle costs, and
        (5) Impact if the current version is not maintained by the supplier or if changes are not able to be incorporated into future versions.

B-2
Note: Modification of COTS software or of open source software managed like COTS software is strongly discouraged.

15. Impacts of upgrades to reusable software products:
   a. Frequency of reusable software product upgrades and modifications, i.e., completely new version or upgrade patches to the existing version being released, being made by its supplier after a particular version has been incorporated into the system;
   b. Feasibility and difficulty of incorporating the new version of the reusable software product into the system;
   c. Impact if the new version is not incorporated (e.g., loss of support); and
   d. Ability of the new architecture to support the evolution of reusable software products.

16. Compatibility of planned upgrades of reusable software products with software development plans and schedules:
   a. Compatibility of planned upgrades with build content and schedules,
   b. Impact on development costs and schedule to incorporate upgrades,
   c. Dependencies among reusable software products:
      (1) Potential for an incompatible set of reusable software products, and
      (2) Potential for schedule delays until all dependent reusable software products are upgraded.

17. Criticality of the functionality provided by the reusable software product:
   a. Appropriateness of the reusable software product to the criticality of the needed functionality, and
   b. Availability of alternate source(s) for the functionality.

18. Short- and long-term cost impacts of using the reusable software product:
   a. Amount of management reserve needed to handle uncertainties (for example, if less of a planned reusable software product is reusable than planned, then more newly developed software is required for that functionality):
      (1) Reusable software product limitations identified, and
      (2) Sufficient management reserve available for contingencies.

19. Technical, cost, and schedule risks and tradeoffs in using the reusable software product:
   a. Ability to tolerate reusable software product problems beyond the project’s control at any point in the system lifecycle, and
   b. Ability to incorporate continuous evolution of reusable software products during development and maintenance.
Appendix C. Discrepancy and Change Reporting

C.1 Scope
This appendix is a MANDATORY part of this standard. This appendix specifies the necessary information for discrepancy and change reports (DCRs), including category classification schemes.

C.2 Discrepancy and Change Reports

C.2.1 Discrepancy and Change Report Definitions

Discrepancy and change reports (DCRs) are recorded for change requests, discrepancies, and test incidents that occur from the beginning of the project through system retirement or project termination.

1. If the developer has alternative definitions for the DCR terms defined in this section, then the developer shall provide the definitions of those terms in the Software Development Plan (SDP).

Terms that apply to discrepancy and change reports (DCRs) are defined below. The definitions are listed in alphabetical order, matching as closely as possible the wording used in Table C.2-5.

Activity where detected. The name of the activity in which the discrepancy or test incident was detected. See Appendix C, Table C.2-2, Categories of Activities, for the activity names in this standard.

Activity where injected. The name of the activity in which the discrepancy was injected. See Appendix C, Table C.2-2, Categories of Activities, for the activity names in this standard.

Affected software and documentation components. The specific software products, documents, paragraphs, files, databases, or any combination to which the report applies.

Analyst assigned. The name of the analyst assigned. (This information is only intended for determining resource loads.)

Artifact types affected. The types of artifacts (e.g., plan, requirements, user guide) affected by the corrective action. See Appendix C, Table C.2-1, Categories of Software Products.

Cause classification. The type of cause of the discrepancy or test incident. For a change request, the cause classification is “enhancement.” See Appendix C, Table C.2-4, Categories of Causes.

Configuration identifier(s). The project-specific configuration identifiers of the affected entities, (e.g., software unit, build, software item, hardware item).

Configuration identifier(s) after corrective action. See Configuration identifier(s) above.

Configuration identifier(s) before corrective action. See Configuration identifier(s) above.

Date analysis completed. The date the analysis was completed.

Date analyst assigned. The date the analyst was assigned.

Date closed. The date that the DCR was closed by the configuration control authority.

Date deployed. The date that the verified implementation solution was deployed at the user site.

Date implementation completed. The date that the implementation of the solution was completed.

Date implementer assigned. The date that the implementer was assigned.

Date of occurrence. The date of the discrepancy or test incident.

Date of origination. The date the DCR is written.
Date solution authorized for deployment. The date the configuration control authority authorized deployment of the implemented solution.

Date solution authorized for implementation. The date the configuration control authority authorized implementation of the solution.

Date verification completed. The date that verification of the solution was completed.

Date verifier assigned. The date that the verifier was assigned to verify the implemented solution.

DCR identifier. The assigned DCR identifier, often generated by the corrective action system.

DCR status. The corrective action system status as defined by the corrective action system procedures.

Description. A description of the change request, discrepancy, or test incident. For discrepancies and test incidents, the description includes what occurred, the conditions and activities leading to the occurrence, the date of occurrence, and the conditions, inputs, and equipment configuration under which the discrepancy or test incident occurred. For discrepancies and test incidents, the description includes sufficient information to permit duplication and analysis. It includes relationships to other reported DCRs and modifications, if any. For change requests, the description includes sufficient information to understand the requested change and its scope.

Description by analyst. See Description above.

Description by originator. See Description above.

Development severity classification. See Appendix C, Table C.2-3, Categories of Severity. The severity of the impact on development if not corrected, (e.g., blocking continued testing, schedule impact). Severity 1 is the highest severity. Contrast with “Operational severity classification” below.

Failure mode classification. The failure category that indicates the way the software performs incorrectly, (e.g., crash, hang, raise exception, issue error message and continue, provide wrong answer and continue, provide information too late to meet requirements). These failure categories are often defined in the project Failure Modes and Effects Analysis (FMEA) or in project corrective action procedures.

Impacts. The cost, schedule, performance, and interface impacts if the solution is approved. Also, the cost, schedule, performance, and interface impacts if the solution is not approved. Include the impact on the other systems, software items, hardware items, other team members, operations, integrated logistics support, system resources, training, and any other applicable impacts.

Implementation solution. A brief description of the implemented solution.

Operating time. Operating time is the interval from the time the software execution is started until the software stops execution due to either a normal exit or an abnormal termination. Operating time units (e.g., hours, minutes, seconds) are also included. Operating times can be aggregated (e.g., if 10 replicas of the item under test are operating simultaneously on different data, or if the operating time is reported as the sum of individual operating times from multiple tests). This item only applies to testing and operations activities.

Operational severity classification. See Appendix C, Table C.2-3, Categories of Severity. The severity of the impact on operations or that would affect operations if not corrected before being installed in the operational environment. Severity 1 is the highest severity. Contrast with Development severity classification above.

Originator. The name, telephone number, email address, and organization of the person submitting the report.
Reason for change. The reason that the change is needed. This item only applies to change requests.

Recommended solution. After analysis of the change request, discrepancy, or test incident, the recommended solution and alternative solutions, if available. When applicable, include supporting rationale and test results.

Short name. A unique, descriptive, short name for the DCR.

Start time. The time at which the software begins execution. Time units are provided, i.e., either wall-clock time, including time zone, or time before or after a significant related event (e.g., installation, launch, start of a mission phase, thruster firing, partial service interruption). If operating time is reported, then this item is optional. If operating time is not reported, then start times are reported for each discrepancy and test incident. This item only applies to testing and operations activities.

Stop time. The time at which the software ends execution due to a normal or abnormal termination. Time units are provided, i.e., either wall-clock time, including the time zone, or time after a significant related event. If operating time is reported, then this item is optional. If operating time is not reported, then stop times are reported for each discrepancy and test incident. This item only applies to testing and operations activities.

System or project name. The name of the system or development project to which the DCR report applies.

Verifier assigned. The name of the verifier assigned. (This information is only intended for determining resource loading.)

Version identifier(s). The software build, or equivalent, version identifiers, individual software unit version identifiers, hardware version identifiers, or other level of version identifiers for the particular software and hardware configurations to which the DCR report applies.

Version identifier(s) before corrective action. See Version identifier(s).

Version identifier(s) after corrective action. See Version identifier(s).

Version identifier(s) after implementation. See Version identifier(s). Note: This item might need updating during verification and for deployment.
Table C.2-1 Categories of Software Products

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Software Product Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>Plans for the project</td>
</tr>
<tr>
<td>b.</td>
<td>Operational concept</td>
</tr>
<tr>
<td>c.</td>
<td>Architecture of the system, subsystem, or software</td>
</tr>
<tr>
<td>d.</td>
<td>Requirements for the system, subsystem, or software</td>
</tr>
<tr>
<td>e.</td>
<td>Design of the system, subsystem, or software</td>
</tr>
<tr>
<td>f.</td>
<td>Software (e.g., code, scripts)</td>
</tr>
<tr>
<td>g.</td>
<td>Database or data file</td>
</tr>
<tr>
<td>h.</td>
<td>Test plan, test case, test procedure, test script, test data</td>
</tr>
<tr>
<td>i.</td>
<td>Test reports</td>
</tr>
<tr>
<td>j.</td>
<td>Manuals and guides, i.e., user, operator, or support manuals or guides</td>
</tr>
<tr>
<td>k.</td>
<td>Standards or policies</td>
</tr>
<tr>
<td>l.</td>
<td>Software development processes or procedures</td>
</tr>
<tr>
<td>m.</td>
<td>Reports other than test reports</td>
</tr>
<tr>
<td>n.</td>
<td>Other software products</td>
</tr>
</tbody>
</table>

Note: This set of categories is called “discrepancy source” in (Abelson 2011).

Table C.2-2 Categories of Activities

<table>
<thead>
<tr>
<th>Section</th>
<th>Activity Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Project planning and oversight</td>
</tr>
<tr>
<td>5.2</td>
<td>Establishing a software development environment</td>
</tr>
<tr>
<td>5.3</td>
<td>System requirements analysis</td>
</tr>
<tr>
<td>5.4</td>
<td>System architecture and design</td>
</tr>
<tr>
<td>5.5</td>
<td>Software requirements analysis</td>
</tr>
<tr>
<td>5.6</td>
<td>Software architecture and design</td>
</tr>
<tr>
<td>5.7</td>
<td>Software implementation and unit testing</td>
</tr>
<tr>
<td>5.8</td>
<td>Unit integration and testing</td>
</tr>
<tr>
<td>5.9</td>
<td>Software item qualification testing</td>
</tr>
<tr>
<td>5.10</td>
<td>Software-hardware item integration and testing</td>
</tr>
<tr>
<td>5.11</td>
<td>System qualification testing</td>
</tr>
<tr>
<td>5.12</td>
<td>Preparing for software transition to operations</td>
</tr>
<tr>
<td>5.13</td>
<td>Preparing for software transition to maintenance</td>
</tr>
<tr>
<td>5.14</td>
<td>Software configuration management</td>
</tr>
<tr>
<td>5.15</td>
<td>Software peer reviews and product evaluations</td>
</tr>
<tr>
<td>5.16</td>
<td>Software quality assurance</td>
</tr>
<tr>
<td>5.17</td>
<td>Corrective action</td>
</tr>
<tr>
<td>5.18</td>
<td>Joint technical and management reviews</td>
</tr>
<tr>
<td>5.19</td>
<td>Software risk management</td>
</tr>
<tr>
<td>5.20</td>
<td>Software measurement</td>
</tr>
<tr>
<td>5.21</td>
<td>Security and privacy</td>
</tr>
<tr>
<td>5.22</td>
<td>Software team member management</td>
</tr>
<tr>
<td>5.23</td>
<td>Interface with software IV&amp;V agents</td>
</tr>
<tr>
<td>5.24</td>
<td>Coordination with associate developers</td>
</tr>
<tr>
<td>5.25</td>
<td>Improvement of project processes</td>
</tr>
</tbody>
</table>
Table C.2-3  Categories of Severity

<table>
<thead>
<tr>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1        | a. Prevents the accomplishment of an operational or essential mission capability  
          b. Jeopardizes safety, security, or other requirement designated “critical” |
| 2        | a. Adversely affects the accomplishment of an operational or essential mission capability, and no workaround solution is known  
          b. Adversely affects technical, cost, or schedule risks to the project or to lifecycle support of the system, and no workaround solution is known |
| 3        | a. Adversely affects the accomplishment of an operational or mission essential capability, but a workaround solution is known  
          b. Adversely affects technical, cost, or schedule risks to the project or to lifecycle support of the system, but a workaround solution is known |
| 4        | a. Results in user or operator inconvenience or annoyance but does not affect a required operational or mission essential capability  
          b. Results in inconvenience or annoyance for development or maintenance personnel, but does not prevent the accomplishment of those responsibilities |
| 5        | Any other effect |

Table C.2-4  Categories of Causes

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Logic</td>
<td>Logic problem</td>
</tr>
<tr>
<td>b. Computational</td>
<td>Computation problem</td>
</tr>
<tr>
<td>c. Interface</td>
<td>Interface problem</td>
</tr>
<tr>
<td>d. Timing</td>
<td>Timing problem</td>
</tr>
<tr>
<td>e. Data handling</td>
<td>Data handling problem, including, e.g., initialization, access, storage, scaling, units</td>
</tr>
<tr>
<td>f. Data problem</td>
<td>Incorrect or missing data</td>
</tr>
<tr>
<td>g. Documentation*</td>
<td>Documentation problem, including, e.g., unclear, missing, conflicting, redundant, confusing, illogical information in documentation</td>
</tr>
<tr>
<td>h. Document quality</td>
<td>Documentation problem (e.g., incomplete information, not meeting standards, not traceable, not current, inconsistencies)</td>
</tr>
<tr>
<td>i. Enhancement</td>
<td>A proposed change for improvement (e.g., new requirement)</td>
</tr>
<tr>
<td>j. Syntax</td>
<td>Nonconformity with defined language rules</td>
</tr>
<tr>
<td>k. Standards</td>
<td>Nonconformity with a defined standard</td>
</tr>
<tr>
<td>l. Other</td>
<td>Other problem</td>
</tr>
</tbody>
</table>

*The Documentation category includes many types of documentation (e.g., requirements, architecture, design, user documentation). The developer may use subcategories of documentation to capture these distinctions.

C.2.2 Discrepancy and Change Report Requirements

This section specifies the requirements for discrepancy and change reports.

1. The developer shall include in each DCR for discrepancies and test incidents, as a minimum, the information specified as “Yes” in the Discrepancy and Test Incident column of Table C.2-5 for the development activities defined in Section 5 of this standard.

   Note: For unit testing, this information is optional.

2. The developer shall include in each DCR for change requests, as a minimum, the information specified as “Yes” in the Change Request column of Table C.2-5 for the development activities defined in Section 5 of this standard.

3. The developer shall record all instances of the same change request, discrepancy, and test incident, if any, in discrepancy and change reports.

4. The developer should provide the solution to the originator for verification.

   Note 1: The developer may record additional DCR information.

   Note 2: The developer may record additional information for any development activity.
<table>
<thead>
<tr>
<th>ID</th>
<th>Discrepancy and Change Report Information Item</th>
<th>Discrepancy and Test Incident</th>
<th>Change Request</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Initiation Information</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td><em>DCR identifier</em></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2.</td>
<td>Short name of DCR</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>3.</td>
<td><em>System or project name</em></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>4.</td>
<td>Originator</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>5.</td>
<td>Date of origination</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>6.</td>
<td>Description by originator</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>7.</td>
<td>Reason for change</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>8.</td>
<td>Operational severity classification</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>9.</td>
<td>Date of occurrence</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>10.</td>
<td>Operating time OR Start time AND Stop time</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note: This information is only required for test and operations activities.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Analysis Information</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>Description by analyst</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>12.</td>
<td>Development severity classification</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>13.</td>
<td>Activity where detected</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>14.</td>
<td>Activity where injected</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>15.</td>
<td>Failure mode classification</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>16.</td>
<td>Artifact types affected</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>17.</td>
<td>Affected <em>software and documentation components</em></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>18.</td>
<td>Configuration identifier(s) before corrective action</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>19.</td>
<td>Version identifier(s) before corrective action</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>20.</td>
<td>Cause classification</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>21.</td>
<td>Recommended solution</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>22.</td>
<td>Impacts</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td><strong>Implementation Information</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23.</td>
<td>Artifact types affected</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>24.</td>
<td>Affected <em>software and documentation components</em>, including file names</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>25.</td>
<td>Configuration identifier(s) after corrective action</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>26.</td>
<td>Version identifier(s) after corrective action</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>27.</td>
<td>Implementation solution</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td><strong>Status Information</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28.</td>
<td><em>DCR status</em></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>29.</td>
<td>Date analyst assigned</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>30.</td>
<td>Analyst assigned</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>31.</td>
<td>Date analysis completed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>32.</td>
<td>Date solution authorized for implementation</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>33.</td>
<td>Date implementer assigned</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ID</td>
<td>Discrepancy and Change Report Information Item</td>
<td>Discrepancy and Test Incident</td>
<td>Change Request</td>
</tr>
<tr>
<td>-----</td>
<td>-------------------------------------------------------------</td>
<td>-------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>34.</td>
<td>Date implementation completed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>35.</td>
<td>Date verifier assigned</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>36.</td>
<td>Verifier assigned</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>37.</td>
<td>Date verification completed**</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>38.</td>
<td>Date solution authorized for deployment</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>39.</td>
<td>Date closed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>40.</td>
<td>Date deployed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* The *developer may have several categories of status, (e.g., open, accepted, rejected, deferred, fixed, verified, closed). The *developer’s DCR status categories are specified in the SDP.

** See Section C.2.2 #4.
Appendix D. Product Evaluations

D.1 Scope
This appendix is a MANDATORY part of this standard. It specifies the requirements and evaluation criteria for product evaluations.

D.2 Criteria Definitions
The following subsections provide definitions for wording used in the evaluation criteria specified in Table D.3-1 that might not be self-explanatory. The criteria are listed in alphabetical order, matching as closely as possible the wording used in Table D.3-1.

1. The definitions provided in Section D.2 shall be used to interpret the evaluation criteria provided in Table D.3-1.
2. If the developer has alternative definitions for the evaluation criteria terms defined in Appendix D, Section D.2, then the developer shall provide the definitions of those terms in the Software Development Plan (SDP).
3. If the developer has additional evaluation criteria planned for use in product evaluations, those criteria shall be defined in the SDP.

D.2.1 Accurately Describes (an Item)
This criterion, applied to user, operator, or programmer instructions and to the “as built” design and version descriptions, means that the instructions or descriptions are correct depictions of the software or other item described.

D.2.2 Adequate Tests, Test Cases, Procedures, Data, and Results
Tests, as described in the Software Test Plan, are adequate if they address all applicable requirements and verification methods and are described in enough detail to guide the development of the test cases for the cited requirements and verification methods. Test cases are adequate if they cover all applicable requirements or design decisions and specify the verification methods, prerequisite conditions, inputs to be used, the expected results, and the criteria to be used for evaluating those results. Test procedures are adequate if they specify the execution steps to be followed in carrying out each test case, expected results, the criteria to be used for evaluating those results, and the steps for analysis. Test data are adequate if they enable the execution of the planned test cases and test procedures. Test or dry run results are adequate if they describe the results of all test cases and show that all criteria have been met, possibly after revision and retesting.

D.2.3 Component-Based
This criterion means that the software architecture consists of components with well-defined interface and service semantics that operate over an underlying infrastructure. The software architecture documents the software components, their semantics, the interfaces, including data and control, among them, and external software-software and software-hardware interfaces.

D.2.4 Consistent with Indicated Product(s)
This criterion means that:
1. No statement or representation in one product contradicts a statement or representation in the other products;
2. A given term, acronym, or abbreviation means the same thing in all of the products; and
3. A given item or concept is referred to by the same name or description in all of the products.
D.2.5 Contains All Applicable Information
This criterion uses the product contents specified in this standard or in the SDP if the product content is not specified in this standard for the required contents of the products that are not deliverable. This criterion uses the Data Item Descriptions (DIDs) or templates, as tailored for the contract, to specify the required content of the products. (See Section 6.2 for a list of the DID and template identifiers for product contents.) The formatting specified in the DID, i.e., required structure and numbering, is not relevant to this evaluation. The SDP describes the artifacts and other information to be developed and recorded for each product required by this standard (see Section 5.1.1).

D.2.6 Covers (a Given Set of Items)
A product “covers” a given set of items if every item in the set has been adequately dealt with in the product. For example, a plan covers the Statement of Work (SOW) if every provision in the SOW is addressed in the plan; a design covers a set of requirements if the design enables the satisfaction of every requirement in the set; a test plan covers a set of requirements if every requirement is the subject of one or more tests that will verify the requirement. “ Covers” is related to the bidirectional traceability (for example, from requirements to design and back) in the requirements, architecture, design, and test planning and test description products.

D.2.7 Feasible
This criterion means that, based upon the knowledge and experience of the evaluator, a given concept, architecture, set of requirements, designs, tests, etc. violates no known principles or lessons learned that would render it impossible to carry out.

D.2.8 Follows SDP
This criterion means that the software product shows evidence of having been developed in accordance with the approach described in the SDP and its updates. Examples include following design and coding standards specified in the plan. For the SDP itself, this criterion applies to updates to the initial plan.

D.2.9 Includes Multiple Perspectives
This criterion means that the software architecture includes multiple perspectives, including both models and detailed textual descriptions of the logical organization; dynamic behavior; process decomposition, including any process priorities, scheduling dependencies, and workflow rates; software organization; physical realization of the software; and data model.

D.2.10 Internally Consistent
This criterion means that:
1. No two statements or representations in a product contradict one another;
2. A given term, acronym, or abbreviation means the same thing throughout the product; and
3. A given item or concept is referred to by the same name or description throughout the product.

D.2.11 Levels of Detail
This criterion means that the architecture or design representation shows high-level components and interfaces as well as other lower-level components and interfaces that transition to the software design.
Note: The level of detail evolves over time from high-level software architecture components and interfaces to multiple lower-level components and interfaces that transition to the software design.

D.2.12 Meets SOW and Contract, if Applicable
This criterion means that the product fulfills all SOW and contract provisions, if any, regarding it. For example, the SOW might place constraints on the operational concept or the design. The contract
might have special provisions concerning software architecture evaluations, process appraisals, data rights, or other standards that contain software requirements.

**D.2.13 Presents a Sound Approach**
This criterion means that, based on the knowledge and experience of the evaluator, a given plan represents a reasonable way to carry out the required activities.

**D.2.14 Shows Evidence that an Item Under Test Meets its Requirements**
This criterion means that recorded test results show that the item under test either passed all tests the first time or was revised and retested until the tests were passed.

**D.2.15 Testable**
A requirement or set of requirements is considered to be testable if an objective and feasible test using one or more of the four verification methods, i.e., Inspection, Analysis, Demonstration, and Test, can be designed to determine whether each requirement has been met.

**D.2.16 Understandable**
This criterion means “understandable by the intended audience.” For example, software products intended for programmer-to-programmer communication need not be understandable by nonprogrammers. A product that correctly identifies its audience and is considered understandable to that audience meets this criterion.

**D.2.17 Uses Software Engineering Tools and Techniques**
This criterion means that the software product was produced using software engineering tool(s) and techniques for representing, documenting, and analyzing the product, including consistency analysis, and traceability mapping. For architecture, this includes using graphical architecture modeling techniques, e.g., Unified Modeling Language (UML).

**D.3 Required Evaluations**
Table D.3-1 specifies the products for which product evaluations are to be performed and the evaluation criteria to be used in those product evaluations. This table is a mandatory part of this standard (see Section 5.15.2.1 for the requirements that reference this table). If a product is not required to be delivered, the product evaluation is still required. However, if the development of a given product has been tailored out of the standard, the requirement to evaluate that product does not apply. Appendix D, Section D.2 provides the definitions of terms used in Table D.3-1.

Each product and criterion is labeled for purposes of identification and tailoring. For convenience, they can be treated as subsections of this section (referring to the first product and its first criterion, for example, as D.3.1.a.). The product names are expressed in lowercase letters to convey that they are generic products, not necessarily in the form of hard-copy documents or deliverable products. Evaluations of system-level products are interpreted as participation in these evaluations. Because some of the criteria are subjective, there is no requirement to prove that the criteria have been met; the requirement is to perform the evaluations using these criteria and to identify possible discrepancies for discussion and resolution.
### Table D.3-1  Products and Associated Evaluation Criteria

<table>
<thead>
<tr>
<th>ID</th>
<th>Product</th>
<th>Evaluation Criteria</th>
</tr>
</thead>
</table>
| 1. | Software development plan §§ 5.1.1, 5.1.6   | a. Contains all *applicable* information in SDP template (see Appendix H.1).  
  b. Meets SOW, if *applicable*.  
  c. Meets *contract*, if *applicable*.  
  d. Understandable.  
  e. Internally consistent.  
  f. Follows SDP for updates to the plan.  
  g. Covers all activities and deliverable *requirements* in the SOW and the SDP CDRL item.  
  h. Covers all activities and *products* in this standard.  
  i. Consistent with other project plans.  
  j. Presents a sound approach to the development.  |
| 2. | Software master build plan §§ 5.1.2.1, 5.6.1, 5.6.2 | a. Contains all *applicable* information in SMBP template (see Appendix H.3).  
  b. Meets SOW, if *applicable*.  
  c. Meets *contract*, if *applicable*.  
  d. Understandable.  
  e. Internally consistent.  
  f. Follows SDP.  
  g. Covers all software *build*-related activities and deliverable *requirements* in the SOW and the SMBP CDRL item, if *applicable*.  
  h. Covers all software *requirements* and *software units*.  
  i. Consistent with the software *requirements*, software *architecture*, and software *design*.  
  j. Presents a sound approach to the integration and delivery.  |
| 3. | Software test plan §§ 5.1.2.2, 5.1.6         | a. Contains all *applicable* information in STP DID.  
  b. Meets SOW, if *applicable*.  
  c. Meets *contract*, if *applicable*.  
  d. Understandable.  
  e. Internally consistent.  
  f. Follows SDP.  
  g. Covers all software-related qualification activities and deliverable *requirements* in the SOW and the STP CDRL item, if *applicable*.  
  h. Covers all *requirements* for the *software items* under *test*.  
  i. Consistent with other project plans.  
  j. Presents a sound approach to the *testing*.  
  k. Contains adequate *tests*.  
  l. Contains adequate plans for a test environment that is sufficiently representative of operations, adequate for the full workload on the environment, and appropriately validated.  |
| 4. | System test plan for a software-only system §§ 5.1.3, 5.1.6 | a. Contains all *applicable* information in STP DID (for a software-only system) or system test plan DID.  
  b. Meets SOW, if *applicable*.  
  c. Meets *contract*, if *applicable*.  
  d. Understandable.  
  e. Internally consistent.  
  f. Follows SDP.  
  g. Covers all software-related qualification activities and deliverable *requirements* in the SOW and the STP or system test plan CDRL item, if *applicable*.  |
<table>
<thead>
<tr>
<th>ID</th>
<th>Product</th>
<th>Evaluation Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>h. Covers all system requirements for the items under test.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i. Consistent with other project plans.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j. Presents a sound approach to the testing.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>k. Contains adequate tests.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>l. Contains adequate plans for a test environment that is sufficiently representative of operations, adequate for the full workload on the environment, and appropriately validated.</td>
</tr>
<tr>
<td>5.</td>
<td>Software installation plan</td>
<td>a. Contains all applicable information in SIP DID.</td>
</tr>
<tr>
<td></td>
<td>§§ 5.1.4, 5.1.6</td>
<td>b. Meets SOW, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d. Understandable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e. Internally consistent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f. Follows SDP.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g. Covers all user site installation activities and deliverable requirements in the SOW and the SIP CDRL item, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h. Consistent with other project plans.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i. Presents a sound approach to the installation.</td>
</tr>
<tr>
<td>6.</td>
<td>Software transition plan</td>
<td>a. Contains all applicable information in STrP DID.</td>
</tr>
<tr>
<td></td>
<td>§§ 5.1.5, 5.1.6</td>
<td>b. Meets SOW, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d. Understandable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e. Internally consistent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f. Follows SDP.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g. Covers all transition-related activities and deliverable requirements in the SOW and the STrP CDRL item, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h. Consistent with other project plans.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i. Presents a sound approach to the transition.</td>
</tr>
<tr>
<td>7.</td>
<td>Operational concept description</td>
<td>a. Contains all applicable information in OCD DID.</td>
</tr>
<tr>
<td></td>
<td>§ 5.3.2</td>
<td>b. Meets SOW, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d. Understandable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e. Internally consistent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f. Follows systems engineering management plan and SDP, as applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g. Covers all software-related operations concept activities and deliverable requirements in the SOW and the OCD CDRL item, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h. Feasible.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i. Consistent with user’s concept of operations and acquirer’s technical requirements document.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j. Consistent with developer’s SSS and SSDD.</td>
</tr>
<tr>
<td>8.</td>
<td>System and subsystem requirements</td>
<td>a. Contains all applicable information in SSS and IRS DIDs.</td>
</tr>
<tr>
<td></td>
<td>§§ 5.3.3, 5.13.7</td>
<td>b. Meets SOW, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d. Understandable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e. Internally consistent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f. Follows system engineering management plan and SDP, as applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g. Covers all system and subsystem requirements activities and deliverable requirements in the SOW and the SSS and IRS CDRL items, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h. Feasible.</td>
</tr>
<tr>
<td>ID</td>
<td>Product</td>
<td>Evaluation Criteria</td>
</tr>
<tr>
<td>----</td>
<td>---------</td>
<td>---------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i. Covers the parent or predecessor requirements allocated to the system or subsystem.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j. Consistent with the user’s concept of operations and acquire’s technical requirements document.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>k. Consistent with the developer’s operational concept.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>l. Consistent with the developer’s system and subsystem architecture.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m. Testable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n. Specifies the verification method for each requirement.</td>
</tr>
<tr>
<td>9.</td>
<td>System-wide architectural design decisions</td>
<td>a. Contains all applicable information in SSDD Did.</td>
</tr>
<tr>
<td></td>
<td>§ 5.4.1</td>
<td>b. Meets SOW, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d. Understandable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e. Internally consistent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f. Follows system engineering management plan and SDP, as applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g. Covers all system-wide architecture design activities and deliverable requirements in the SOW and SSDD CDRL item, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h. Consistent with the system requirements, including interface and database requirements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i. Consistent with the user’s concept of operations and developer’s operations concept description.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j. Feasible.</td>
</tr>
<tr>
<td>10.</td>
<td>System architectural design</td>
<td>a. Contains all applicable information in SSDD Did.</td>
</tr>
<tr>
<td></td>
<td>§ 5.4.2</td>
<td>b. Meets SOW, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d. Understandable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e. Internally consistent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f. Follows system engineering management plan and SDP, as applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g. Covers all system architectural design activities and deliverable requirements in the SOW and SSDD CDRL item, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h. Covers the system and subsystem requirements, including interface and database requirements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i. Consistent with the system-wide design decisions, system and subsystem requirements, user’s concept of operations, and developer’s operations concept description.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j. Feasible.</td>
</tr>
<tr>
<td>11.</td>
<td>Software requirements and software interface requirements</td>
<td>a. Contains all applicable information in SRS and IRS DiDs.</td>
</tr>
<tr>
<td></td>
<td>§§ 5.5, 5.13.6</td>
<td>b. Meets SOW, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d. Understandable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e. Internally consistent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f. Follows SDP.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g. Covers all software requirements-related activities and deliverable requirements in the SOW and the SRS and IRS CDRL items, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h. Covers all system and subsystem requirements allocated to the software item.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i. Feasible.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j. Testable.</td>
</tr>
<tr>
<td>ID</td>
<td>Product</td>
<td>Evaluation Criteria</td>
</tr>
<tr>
<td>----</td>
<td>---------</td>
<td>---------------------</td>
</tr>
<tr>
<td>k.</td>
<td>Each requirement, including interface and derived requirements, has been bidirectionally traced to its parent and back.</td>
<td></td>
</tr>
<tr>
<td>l.</td>
<td>Each requirement has its verification method(s) specified.</td>
<td></td>
</tr>
<tr>
<td>m.</td>
<td>All requirements have been reviewed, and any discrepancies have been reported and corrected.</td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>Overall software architecture</td>
<td>a. Contains all applicable information in SAD template (see Appendix H.2).</td>
</tr>
<tr>
<td>§§ 5.6.1, 5.13.4</td>
<td>b. Meets SOW, if applicable.</td>
<td>c. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td>d. Understandable.</td>
<td>e. Internally consistent.</td>
</tr>
<tr>
<td></td>
<td>f. Follows SDP.</td>
<td>g. Covers all software architecture activities and deliverable requirements in the SOW and SAD CDRL items, if applicable.</td>
</tr>
<tr>
<td></td>
<td>h. Covers all software requirements, including interface requirements.</td>
<td>i. Consistent with system and subsystem requirements.</td>
</tr>
<tr>
<td></td>
<td>j. Consistent with overall software requirements.</td>
<td>k. The modeling analysis shows evidence the architecture will meet the required margin reserves of the target computer system(s) selected (see Section 4.2.6).</td>
</tr>
<tr>
<td></td>
<td>l. Able to meet each software item's allocated computer resource reserve margin requirements, given the selected computer hardware and the software and interface requirements to be met.</td>
<td>m. Feasible.</td>
</tr>
<tr>
<td>n.</td>
<td>COTS software trial results and product and vendor viability reports reviewed and risk mitigation plans updated.</td>
<td></td>
</tr>
<tr>
<td>o.</td>
<td>COTS software and any other reusable software are isolated for possible later substitution.</td>
<td></td>
</tr>
<tr>
<td>p.</td>
<td>Rationale for architecture decisions included, including any current technology constraints.</td>
<td></td>
</tr>
<tr>
<td>q.</td>
<td>Component-based.</td>
<td></td>
</tr>
<tr>
<td>r.</td>
<td>Consistent with system and subsystem architecture and design.</td>
<td></td>
</tr>
<tr>
<td>s.</td>
<td>Includes multiple architecture perspectives.</td>
<td></td>
</tr>
<tr>
<td>t.</td>
<td>Uses software engineering tools and techniques.</td>
<td></td>
</tr>
<tr>
<td>u.</td>
<td>Uses graphical architecture modeling techniques, e.g., Unified Modeling Language (UML).</td>
<td></td>
</tr>
<tr>
<td>v.</td>
<td>Contains an appropriate level of detail.</td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td>Software item architecture</td>
<td>a. Contains all applicable information in SAD template (see Appendix H.2).</td>
</tr>
<tr>
<td>§§ 5.6.2, 5.13.4</td>
<td>b. Meets SOW, if applicable.</td>
<td>c. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td>d. Understandable.</td>
<td>e. Internally consistent.</td>
</tr>
<tr>
<td></td>
<td>f. Follows SDP.</td>
<td>g. Covers all software item architecture activities and deliverable requirements in the SOW and SAD CDRL items, if applicable.</td>
</tr>
<tr>
<td></td>
<td>h. Covers all software item requirements, including interface requirements.</td>
<td>i. Consistent with overall software requirements, including interface requirements.</td>
</tr>
<tr>
<td>ID</td>
<td>Product</td>
<td>Evaluation Criteria</td>
</tr>
<tr>
<td>----</td>
<td>---------</td>
<td>---------------------</td>
</tr>
</tbody>
</table>
| 14 | Software item detailed design | a. Contains all applicable information in SDD, IDD, and DBDD DIDs.  
| §§ 5.6.3, 5.13.4 | b. Meets SOW, if applicable.  
| | | c. Meets contract, if applicable.  
| | | d. Understandable.  
| | | e. Internally consistent.  
| | | f. Follows SDP, including design standards.  
| | | g. Covers all software design activities and deliverable requirements in the SOW and software, interface, and database design CDRL items, if applicable.  
| | | h. Covers all software item requirements allocated to each unit.  
| | | i. Consistent with software item architecture.  
| | | j. Consistent with software item-wide design decisions.  
| | | k. Feasible for design, implementation, operations, and maintenance.  
| | | l. Testable.  
| | | m. Consistent with possibly updated software estimates.  
| | | n. COTS software and any other reusable software product trial results along with product and vendor viability reports reviewed and risk mitigation plans updated.  
| | | o. COTS software and other reusable software isolated for possible later substitution.  
| | | p. Design modifications of reusable software necessary, correct, and complete for satisfying the requirements allocated to this reuse, and software estimates updated accordingly.  

<table>
<thead>
<tr>
<th>ID</th>
<th>Product</th>
<th>Evaluation Criteria</th>
</tr>
</thead>
</table>
| | | j. Consistent with system and subsystem requirements.  
| | | k. Consistent with overall software architecture and design decisions.  
| | | l. Consistent with system and subsystem architecture and design.  
| | | m. Able to meet the software item’s allocated computer resource reserve margin requirements, given the selected computer hardware and the software and interface requirements to be met.  
| | | n. Feasible.  
| | | o. Testable.  
| | | p. COTS software trial results and product and vendor viability reports reviewed and risk mitigation plans updated.  
| | | q. COTS software and any other reusable software are isolated for possible later substitution.  
| | | r. Rationale for architecture decisions is included, including any current technology constraints.  
| | | s. Component-based.  
| | | t. Includes multiple architecture perspectives.  
| | | u. Uses software engineering tools and techniques.  
| | | v. Uses graphical architecture modeling techniques, e.g., Unified Modeling Language (UML).  
| | | w. Contains an appropriate level of detail.  

D-8
<table>
<thead>
<tr>
<th>ID</th>
<th>Product</th>
<th>Evaluation Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Software implementation, i.e., code and other necessary files, e.g., data files, database files, build files</strong></td>
<td>§ 5.7.1</td>
</tr>
<tr>
<td>15</td>
<td>a. Meets SOW, if applicable.</td>
<td>b. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td>e. Follows SDP, including coding standards.</td>
<td>f. Covers the software item detailed design.</td>
</tr>
<tr>
<td></td>
<td>g. Consistent with software architecture and software item-wide design decisions.</td>
<td>h. Feasible for operations and maintenance.</td>
</tr>
<tr>
<td></td>
<td>i. Testable.</td>
<td>j. Algorithms properly implemented.</td>
</tr>
<tr>
<td></td>
<td>k. COTS software and other reusable software isolated for possible later substitution.</td>
<td>l. Follows coding standards that do not allow dangerous practices and that enhance dependability, as applicable to the software language and category of software.</td>
</tr>
<tr>
<td></td>
<td>m. Analyzed using static and dynamic code analysis tools to ensure coding standards are met and no vulnerabilities exist.</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td><strong>Software unit testing test cases, procedures, and results</strong></td>
<td>§§ 5.7.2, 5.7.3, 5.7.4, 5.7.5, 5.7.6</td>
</tr>
<tr>
<td></td>
<td>a. Meets SOW, if applicable.</td>
<td>b. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td>e. Follows SDP.</td>
<td>f. Covers all units in the software item detailed design.</td>
</tr>
<tr>
<td></td>
<td>g. Test cases, procedures, data, and results provide evidence that the units fully and correctly implement the design and satisfy their allocated requirements.</td>
<td>h. Satisfies the requirements of Section 5.7.</td>
</tr>
<tr>
<td></td>
<td>i. Test cases, procedures, data, and results are captured in SDFs.</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td><strong>Software integration testing test cases, procedures, and results</strong></td>
<td>§§ 5.8.1 through 5.8.6</td>
</tr>
<tr>
<td></td>
<td>a. Meets SOW, if applicable.</td>
<td>b. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td>e. Follows SDP.</td>
<td>f. Consistent with the integration sequence in the SMBP.</td>
</tr>
<tr>
<td></td>
<td>g. Covers the integration of all software in the software item architecture and detailed design, including all reusable software, modified reusable software, and newly developed software.</td>
<td>h. Test cases, procedures, data, and results provide evidence that the integrated software fully and correctly implements the design and satisfies its allocated requirements.</td>
</tr>
<tr>
<td></td>
<td>i. Satisfies the requirements of Section 5.8.</td>
<td>j. Test cases, procedures, data, and results are captured in SDFs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td><strong>Software item qualification test descriptions</strong></td>
<td>§§ 5.9.3, 5.9.8</td>
</tr>
<tr>
<td></td>
<td>a. Contains all applicable information in STD DID.</td>
<td>b. Meets SOW, if applicable.</td>
</tr>
<tr>
<td></td>
<td>c. Meets contract, if applicable.</td>
<td>d. Understandable.</td>
</tr>
<tr>
<td></td>
<td>e. Internally consistent.</td>
<td>f. Follows SDP.</td>
</tr>
<tr>
<td></td>
<td>g. Covers all planned software qualification tests.</td>
<td>h. Covers all software item and software interface requirements.</td>
</tr>
<tr>
<td></td>
<td>i. Consistent with the software test plan.</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Product</td>
<td>Evaluation Criteria</td>
</tr>
<tr>
<td>----</td>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>j.</td>
<td>Contains test cases, procedures, databases, and data that will enable the software requirements to be verified.</td>
<td></td>
</tr>
<tr>
<td>k.</td>
<td>Contains an appropriate level of detail for the test procedures to be repeatable.</td>
<td></td>
</tr>
<tr>
<td>l.</td>
<td>Test cases are consistent with the verification methods and levels in the software and interface requirements specifications and in the software test plan.</td>
<td></td>
</tr>
<tr>
<td>m.</td>
<td>Software and software interface requirements are allocated to test procedure steps that in combination completely cover the requirements.</td>
<td></td>
</tr>
<tr>
<td>n.</td>
<td>Feasible.</td>
<td></td>
</tr>
<tr>
<td>o.</td>
<td>Covers all of the software item verification requirements in Section 5.9.</td>
<td></td>
</tr>
<tr>
<td>19.</td>
<td>Software item qualification test results § 5.9.6 See § 3.1 for definition of verification method and definitions of A = Analysis, D = Demonstration, I = Inspection, and T = Test</td>
<td>a. Contains all applicable information in STR DID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b. Meets SOW, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c. Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d. Understandable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e. Internally consistent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f. Follows SDP.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g. Covers all planned software item qualification test cases.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h. Covers all planned software item and software interface requirements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i. Contains as run redlined test procedures, with software quality assurance witness signature or stamp indicating that the test procedures were executed as redlined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j. Contains complete documentation of all test results and all post-test analyses performed for requirements with verification methods Demonstration and Test.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>k. Contains complete analysis for all requirements with verification method Analysis and complete evidence that the Inspection was performed for all requirements with verification method Inspection.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>l. Shows evidence that the software item meets all of its functional, performance, and other nonfunctional requirements, using both nominal and off-nominal scenarios.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m. Shows evidence that all discrepancies detected during software item qualification dry runs and formal execution have been recorded and entered into the discrepancy reporting system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n. Shows evidence that the verification status of the requirements under test is maintained in an enduring manner and that the status accurately reflects the results of the software item qualification testing.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o. Shows evidence that the test environment configuration used for the formal execution is consistent with that described in the test procedures before the testing began, and if there are differences between the as run test environment configuration and that described in the test procedure, then all differences are explained and shown not to affect the verification of the requirements.</td>
</tr>
<tr>
<td>ID</td>
<td>Product</td>
<td>Evaluation Criteria</td>
</tr>
<tr>
<td>----</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 20 | Software-hardware integration testing test cases, procedures, and results | a. Meets SOW, if applicable.  
b. Meets contract, if applicable.  
c. Understandable.  
d. Internally consistent.  
e. Follows systems engineering management plan and SDP, as applicable.  
f. Consistent with the software and hardware integration sequence in the SMBP.  
g. Covers the integration of all software and target computer system hardware in the software architecture.  
h. Contains test cases, procedures, databases, and data that enable the software-hardware interface requirements to be verified.  
i. Test cases, procedures, data, and results provide evidence that the integrated software and target computer system hardware fully and correctly implement the design and satisfy their allocated requirements.  
j. Satisfies the requirements of Section 5.10.  
k. Test cases, procedures, data, and results are captured in SDFs.                                                                                      |
| 21 | System and subsystem qualification test descriptions                      | a. Contains all applicable information in the system and subsystem test description DID (or STD DID for a software-only system).  
b. Meets SOW, if applicable.  
c. Meets contract, if applicable.  
d. Understandable.  
e. Internally consistent.  
f. Follows systems engineering management plan and SDP, as applicable.  
g. Covers all planned system and subsystem qualification tests.  
h. Covers all planned system and subsystem requirements, including interface requirements.  
i. Consistent with the system and subsystem test plan(s).  
j. Contains test cases, procedures, databases, and data that enable the system and subsystem requirements, including interface requirements, to be verified.  
k. Contains an appropriate level of detail for the test procedures to be repeatable.  
l. Test cases are consistent with the verification methods and levels in the system and subsystem requirements specification(s) and in the system and subsystem test plan(s).  
m. System and subsystem requirements, including interface requirements, are allocated to test procedure steps that in combination completely cover the requirements.  
n. Feasible.                                                                                                                                           |
<table>
<thead>
<tr>
<th>ID</th>
<th>Product</th>
<th>Evaluation Criteria</th>
</tr>
</thead>
</table>
| 22 | System and subsystem qualification test results             | a. Contains all applicable information in system and subsystem test report DID (or STR DID for a software-only system).  
b. Meets SOW, if applicable.  
c. Meets contract, if applicable.  
d. Understandable.  
e. Internally consistent.  
f. Follows systems engineering management plan and SDP, as applicable.  
g. Covers all planned system and subsystem qualification test cases.  
h. Covers all planned system and subsystem requirements, including interface requirements.  
i. Contains as run redlined test procedures, with quality assurance witness signature or stamp indicating that the test procedures were executed as redlined.  
j. Contains complete documentation of all test results and all post-test analysis performed for requirements with verification methods Demonstration and Test.  
k. Contains complete analysis for all requirements with verification method Analysis and complete evidence that the Inspection was performed for all requirements with verification method Inspection.  
l. Shows evidence that the system and subsystem meet all of their functional, performance, and other nonfunctional requirements, using both nominal and off-nominal scenarios.  
m. Shows evidence that all discrepancies detected during system and subsystem qualification dry runs and formal execution have been recorded and entered into the discrepancy reporting system.  
n. Shows evidence that the verification status of the requirements under test is maintained in an enduring manner and that the status accurately reflects the results of the system and subsystem qualification testing.  
o. Shows evidence that the test environment configuration used for the formal execution is consistent with that described in the test procedures before the testing began, and if there are differences between the as run test environment configuration and that described in the test procedure, then all differences are explained and shown not to affect the verification of the requirements.  |
| 23 | Executable software                                         | a. Meets all requirements in Section 3.1 of SPS DID.  
b. Meets SOW, if applicable.  
c. Meets contract, if applicable.  
d. Understandable.  
e. Internally consistent.  
f. Follows SDP.  
g. Meets delivery requirements.  
h. Contains all software necessary for execution.  
i. Shows evidence that the version of the executable software exactly matches the version that passed testing.  
j. Deliverable media accurately labeled.  
k. Shows evidence that the version of the executable software was created from the specific identified configuration-managed version of the source code.  |
<table>
<thead>
<tr>
<th>ID</th>
<th>Product</th>
<th>Evaluation Criteria</th>
</tr>
</thead>
</table>
| 24. | Software version descriptions | a. Contains all *applicable* information in SVD *DID*.  
     | §§ 5.12.2, 5.13.3 | b. Meets *SOW*, if *applicable*.  
     |                                   | c. Meets *contract*, if *applicable*.  
     |                                   | d. Understandable.  
     |                                   | e. Internally consistent.  
     |                                   | f. Follows SDP.  
     |                                   | g. Meets delivery *requirements*.  
     |                                   | h. Accurately identifies the version of each *software component*, i.e., file, unit, *software item*, delivered.  
     |                                   | i. Accurately identifies the changes incorporated.  
     |                                   | j. Accurately identifies the known *discrepancies*, errors, error avoidance, liens, and workarounds.  
     |                                   | k. Provides accurate installation instructions for each *user site*. |
| 25. | Software user manuals | a. Contains all *applicable* information in SUM *DID*.  
     | § 5.12.3.1 | b. Meets *SOW*, if *applicable*.  
     |                                   | c. Meets *contract*, if *applicable*.  
     |                                   | d. Understandable.  
     |                                   | e. Internally consistent.  
     |                                   | f. Follows SDP.  
     |                                   | g. Accurately describes software installation and use to the intended audience of this manual. |
     | § 5.12.3.2 | b. Meets *SOW*, if *applicable*.  
     |                                   | c. Meets *contract*, if *applicable*.  
     |                                   | d. Understandable.  
     |                                   | e. Internally consistent.  
     |                                   | f. Follows SDP.  
     |                                   | g. Accurately describes the operational characteristics of the *target computer system* and procedures for its operation.  
     |                                   | h. Does not duplicate information available in commercial manuals. |
| 27. | Source files | a. Contains all *applicable* information in SPS *DID*.  
     | § 5.13.2 | b. Meets *SOW*, if *applicable*.  
     |                                   | c. Meets *contract*, if *applicable*.  
     |                                   | d. Understandable.  
     |                                   | e. Internally consistent.  
     |                                   | f. Follows SDP.  
     |                                   | g. Meets delivery *requirements*.  
     |                                   | h. All required *software* is present.  
     |                                   | i. Shows evidence that the version of the *source code* exactly matches the version that passed *testing*.  
     |                                   | j. Deliverable media accurately labeled.  
     |                                   | k. Shows evidence that *source code* exactly matches the specific identified configuration-managed version of the *source code*.  
     |                                   | l. Consistent with the *software units* and data in the "as built" *software design*. |
| 28. | "As built" software item design and related information | a. Contains all *applicable* information in SPS *DID*.  
     | § 5.13.4 | b. Meets *SOW*, if *applicable*.  
     |                                   | c. Meets *contract*, if *applicable*.  
     |                                   | d. Understandable.  
     |                                   | e. Internally consistent.  
     |                                   | f. Follows SDP.  
<pre><code> |                                   | g. Accurately describes the &quot;as built&quot; design of the *software item*. |
</code></pre>
<table>
<thead>
<tr>
<th>ID</th>
<th>Product</th>
<th>Evaluation Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Product Evaluation Criteria</td>
<td></td>
</tr>
<tr>
<td></td>
<td>h. Accurately describes compilation and build procedures.</td>
<td>i. Accurately describes modification procedures.</td>
</tr>
<tr>
<td>29.</td>
<td>“As built” system and subsystem design</td>
<td>a. Contains all applicable information in SSDD DID.</td>
</tr>
<tr>
<td></td>
<td>§ 5.13.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>§ 5.13.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>§§ 5.13.8.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>§§ 5.13.8.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32.</td>
<td>Software measurement plan</td>
<td>a. Contains all applicable information in SMP template (see Appendix H.4).</td>
</tr>
<tr>
<td></td>
<td>§§ 5.20.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33.</td>
<td>Software measurement report</td>
<td>a. Contains all applicable information in SMR template (see Appendix H.5).</td>
</tr>
<tr>
<td></td>
<td>§§ 5.20.2</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Product</td>
<td>Evaluation Criteria</td>
</tr>
<tr>
<td>----</td>
<td>---------</td>
<td>---------------------</td>
</tr>
<tr>
<td></td>
<td>e.</td>
<td>Internally consistent.</td>
</tr>
<tr>
<td></td>
<td>f.</td>
<td>Follows SDP.</td>
</tr>
<tr>
<td></td>
<td>g.</td>
<td>Covers all required software measures.</td>
</tr>
<tr>
<td></td>
<td>h.</td>
<td>Covers all software measurement-related activities in the SOW and the SDP, SMP, and SMR CDRL items.</td>
</tr>
<tr>
<td></td>
<td>i.</td>
<td>Consistent with all software requirements, builds, architecture, test plans, critical computer resources, and TPMs.</td>
</tr>
<tr>
<td></td>
<td>j.</td>
<td>Provides insight into development with current data and explanations.</td>
</tr>
<tr>
<td>34.</td>
<td>Process improvement plan § 5.25</td>
<td>a.</td>
</tr>
<tr>
<td></td>
<td>b.</td>
<td>Meets SOW, if applicable.</td>
</tr>
<tr>
<td></td>
<td>c.</td>
<td>Meets contract, if applicable.</td>
</tr>
<tr>
<td></td>
<td>d.</td>
<td>Understandable.</td>
</tr>
<tr>
<td></td>
<td>e.</td>
<td>Internally consistent.</td>
</tr>
<tr>
<td></td>
<td>f.</td>
<td>Follows SDP.</td>
</tr>
<tr>
<td></td>
<td>g.</td>
<td>Provides a sound approach to software process improvement on the project.</td>
</tr>
<tr>
<td></td>
<td>h.</td>
<td>Addresses all needed process improvement areas identified by the project's appraisals.</td>
</tr>
<tr>
<td>35.</td>
<td>Sampling of software development files §§ 5.1.2.1, 5.2.4, 5.6.3, 5.7.1, 5.7.2, 5.7.4, 5.7.5, 5.7.6, 5.8.2, 5.8.4, 5.8.5, 5.8.6, 5.9.6, 5.9.7, 5.9.8, 5.10.2, 5.10.4, 5.10.5, 5.10.6, 5.11.3, 5.11.6, 5.11.7, 5.11.8</td>
<td>a.</td>
</tr>
<tr>
<td></td>
<td>b.</td>
<td>Understandable.</td>
</tr>
<tr>
<td></td>
<td>c.</td>
<td>Internally consistent.</td>
</tr>
<tr>
<td></td>
<td>d.</td>
<td>Follows SDP.</td>
</tr>
<tr>
<td></td>
<td>e.</td>
<td>Contents are current with the ongoing effort.</td>
</tr>
<tr>
<td></td>
<td>f.</td>
<td>Adequate unit test cases, procedures, data, and results.</td>
</tr>
<tr>
<td></td>
<td>g.</td>
<td>Adequate unit integration test cases, procedures, data, and results.</td>
</tr>
<tr>
<td></td>
<td>h.</td>
<td>Adequate software item qualification dry run results.</td>
</tr>
<tr>
<td></td>
<td>i.</td>
<td>Adequate software-hardware item integration test cases, procedures, data, and results.</td>
</tr>
<tr>
<td></td>
<td>j.</td>
<td>Adequate system qualification dry run results for software-only systems.</td>
</tr>
</tbody>
</table>
Appendix E. Joint Technical and Management Reviews

E.1 Scope
This appendix is a MANDATORY part of this standard. This appendix describes a set of joint technical reviews and joint management reviews that are held during a software development project.

Note 1: It is acceptable for any of these joint reviews to be held incrementally as agreed to by the acquirer and developer. Examples include covering all of the applicable topics for a build, covering a subset of the applicable topics for one or more software items, or any combination thereof.

Note 2: If a software item is developed in multiple builds, some of the software products will not be completed until the final build. For these reviews, the software products for each build are interpreted to include those parts of software products needed to meet the requirements to be implemented in the build(s) under review.

E.2 Joint Technical Reviews
Joint technical reviews consist of two types: software-specific joint technical reviews (see Section E.3) and joint technical reviews supporting major reviews (see Section E.4). The following requirements supplement the requirements specified in Section 5.18.1 for joint technical reviews.

1. The developer shall provide the acquirer a minimum of two weeks of advance notice of the exact time and location of:
   a. software-specific joint technical reviews, and
   b. joint technical reviews supporting major reviews.

Note: The developer and acquirer can mutually agree upon a different advance notification length of time.

E.3 Software-Specific Joint Technical Reviews
Note 1: If a system or software item is developed in multiple builds, the types of joint technical reviews held and the criteria applied depend on the objectives of each build.

Note 2: This section is written in terms of software build reviews. However, if the system or software item is developed using a once-through i.e., waterfall, lifecycle model, the following requirements apply with the interpretation that the software is developed as a single build.

1. The developer shall hold software-specific joint technical reviews for each build.

2. For each build the developer shall select and propose the software-specific joint technical reviews to be held for that build.

3. For each build the developer shall select from the following software-specific joint technical review candidates:
   a. Software Build Planning Review (SBPR),
   b. Software Build Requirements and Architecture Review (SBRAR),
   c. Software Build Design Review (SBDR),
   d. Software Build Test Readiness Review (SBTRR), and
   e. Software Build Exit Review (SBER).

Note: The developer can propose additional or different software-specific joint technical reviews from the reviews specified above, if justified by the build objectives.

4. The developer shall document the following information in the SDP:
   a. The software-specific joint technical reviews to be held for each build,
   b. The rationale for selecting the software-specific joint technical reviews to be held for each build, and
   c. The objectives for each software-specific joint technical review selected, if different from the objectives specified in this appendix.

Note 1: Different builds can have the same or different objectives for a particular type of build.
software-specific joint technical review. If they differ, the developer should specify the differences.

Note 2: The developer may tailor the build objectives for the reviews specified in this appendix or may define the build objectives for any additional or different software-specific joint technical reviews.

5. For each build, the developer shall define the following in the SDP for each selected software-specific joint technical review:
   a. Entrance criteria,
   b. Exit criteria,
   c. Products to be reviewed, and
   d. Mandatory attendees.

6. For each software-specific joint technical review, the developer shall:
   a. Document the action items generated during each software-specific joint technical review,
   b. Maintain the action item list, and
   c. Track the action items to closure.

   Note: Closure of an action item can result in, for example, the issue being resolved or a discrepancy and change report (DCR) being generated.

7. For each software-specific joint technical review, the developer shall:
   a. Review all open action items from previous reviews, and
   b. Describe plans for closure of those action items that are still open.

8. For each software-specific joint technical review, the developer shall:
   a. Review the objectives and exit criteria of previous software-specific joint technical reviews applicable to the build(s) under review,
   b. Determine whether any changes since the previous software-specific joint technical reviews have caused one or more objectives or exit criteria to no longer be satisfied,
   c. Discuss the impact of any unsatisfied prior objectives and exit criteria, and
   d. Describe plans for returning the unsatisfied prior objectives and exit criteria to a satisfied state.

   Note: Of particular importance are the objectives and exit criteria of the SBPR, since that review examines whether the allocated resources are sufficient to develop the build content on the required schedule. This resource and schedule issue needs to be re-examined in light of all changes that have happened since the previous SBPR.

9. The developer shall capture lessons learned to improve processes for future software-specific joint technical reviews.

10. The developer shall provide evidence to support the satisfaction of the software-specific joint technical review objectives and exit criteria.

    Note: Evidence, for example, can consist of part or all of the software products themselves, engineering analyses, trade studies, models, and simulations, as well as other types of analyses.

11. Each software-specific joint technical review shall have the following general objectives:
   a. Review evolving software products, using as minimum criteria the software product evaluation criteria in Appendix D;
   b. Review and demonstrate proposed technical solutions;
   c. Provide insight and obtain feedback on the technical effort;
   d. Surface and resolve technical issues;
   e. Surface near- and long-term risks regarding technical, cost, and schedule issues;
   f. Arrive at agreed-upon mitigation strategies for identified risks, within the authority of those present;
   g. Identify risks and issues to be raised at joint management reviews;
   h. Update the risk mitigation plans;
   i. Determine if software risk mitigation is proceeding according to the updated software risk mitigation plans; and
j. Provide for ongoing communication between *acquirer* and *developer* technical personnel.

### E.3.1 Software Build Planning Review (SBPR)

This subsection provides the specific objectives for the Software Build Planning Review (SBPR). All objectives apply to the software *products* as completed for the *build(s)* under review. See Section E.3 for overall *requirements* that apply to all software-specific joint technical reviews.

1. The objectives of the SBPR **shall** be to ensure that:
   a. The *requirements* allocated to *software* and the overall software architectural *design* are sufficiently mature for accurate build planning.
   b. The software *processes*, including standards and procedures:
      (1) Are sufficiently defined, mature, and effective for performing the build *software development* activities;
      (2) Are suitable for the project scope and complexity;
      (3) Include lessons learned from use on previous *builds*; and
      (4) Are *documented* in the Software Development Plan.
   c. The overall build plan is:
      (1) Suitable for the project scope and complexity;
      (2) Suitable for meeting the needs and timing of the higher integration levels; and
      Note: Higher integration levels include, e.g., hardware-software integration, *subsystem* integration, and *system* integration.
      (3) Suitable for meeting the needs and timing of *contract* milestones.
      Note: Examples of software-related *contract* milestones are external interface *tests*, installation in *user sites*, and *turnover* to operations.
   d. The build content for the *build(s)* under review is clearly *defined* and *documented*, including:
      (1) The *requirements* allocated to the *build*;
      (2) The functions to be implemented in the *build*;
      (3) The architecture and design *components* to be implemented in the *build*, and
      (4) The *discrepancy and change reports* to be fixed in the *build*.
   e. The software *risks* for the *build(s)* under review:
      (1) Are complete and current, as of the time of the review;
      (2) Have well-defined, effective mitigation plans in place; and
      (3) Are being tracked to retirement.
   f. The resources allocated to the *build* are sufficient to develop the build content on its required schedule for each *build* under review.
   g. The impacts from previous *builds* and from external sources upon the *build(s)* under review remain manageable within the allocated build resources and schedule(s), including impacts from:
      (1) Unsatisfied *requirements* reallocated from previous *builds* to the *build(s)* under review;
      (2) Changes to the *requirements* allocated to the *build(s)* under review;
      (3) The *discrepancy and change reports* allocated to the *build(s)* under review;
      (4) Changes to the previously developed software *architecture, design, and code*; and
      (5) Newly identified software *risks* or known *risks* with increased impact or likelihood of occurrence.
   h. The resources allocated for the *build(s)* under review are sufficient to handle:
      (1) Overlapping build development; and
      (2) Support of higher-level integration and operations, as *applicable.*
i. Implementation of the software engineering environment is sufficiently mature to proceed with software and system development activities that will use these facilities, including the following aspects:

1. Software engineering tools for configuration management, requirements, architecture, design, coding, integration, debugging, and development testing are in place, or planned to be in place, by the time they are needed for the build(s) under review; and
2. The software engineering environment, in place or planned to be in place, has sufficient capability and capacity for concurrent build(s).

j. Implementation of the software integration and qualification test environment is sufficiently mature to proceed with software and system development activities that will use these facilities, including the following aspects:

1. The software integration and qualification test environment is in place, or planned to be in place, by the time it is needed for the build(s) under review; and
2. The software integration and qualification test environment, in place or planned to be in place, has sufficient capability and capacity for testing the build(s) under review and the concurrent builds being developed, integrated, and fielded.

k. The SDP and software standards and procedures (e.g., work instructions) are adequate for the software development activities to be performed in the build(s) under review and for the program scope and complexity.

l. Personnel performing the software development activities for the build(s) under review are in place, fully trained, and proficient in the applicable processes, standards, and procedures.

m. The software risks for this build have been updated to include identification and assessment of new risks and reassessment of existing risks:

1. Updated software risk mitigation plans are in place, and
2. Software risk mitigation is proceeding according to the updated software risk mitigation plans.

n. Software size, effort, cost, schedule, and staffing estimates and actuals are realistic and appropriate for the build(s) under review.

o. The software measurements for the previous build(s), if any, are realistic and within acceptable tolerances.

E.3.2 Software Build Requirements and Architecture Review (SBRAR)

This subsection provides the specific objectives for the Software Build Requirements and Architecture Review (SBRAR). All objectives apply to the software products as completed for the build(s) under review. See Section E.3 for overall requirements that apply to all software-specific joint technical reviews.

Note: All of the software requirements are specified in the Software Requirements Specification (SRS) and the Interface Requirements Specification (IRS). (See Section 6.2 for the SRS and IRS DID identifiers.)

1. The objectives of the SBRAR shall be to ensure that:

a. The software requirements, including interface requirements, specialty engineering requirements (e.g., Human Systems Integration, information assurance, dependability), performance requirements, i.e., Key Performance Parameters, response time, and end-to-end timing requirements, are sufficiently mature to proceed with dependent software and system development activities, including the following aspects:

1. Higher level requirements allocated to software are complete and stable;
2. Software requirements are stable, correct, complete, consistent, feasible, verifiable, and clearly and unambiguously stated;
3. Software requirements have all TBXs resolved;
Note: TBXs include To be determined (TBDs), To be resolved (TBRs), and To be satisfied (TBSs).

4. Software requirements include necessary requirements derived from the system and software architecture, system operational concepts, trade studies, and design decisions;

5. Software requirements are fully bidirectionally traced to and from their parent requirements, or to and from the source(s) from which they are derived;

6. Software requirements include requirements for off-nominal and error conditions;

7. Software interface requirements are agreed to by all parties;

8. Software requirements:
   (a) Have one or more valid verification methods and levels specified, and
   (b) Are able to be fully verified by those methods and levels;

9. Each software requirement is allocated to one or more software architecture components; and

10. Software requirements are fully bidirectionally traced to and from the software architecture components by which they will be implemented.

b. The software architecture is sufficiently mature to proceed with dependent software and system development activities, including the following aspects:

   1. The software architecture is complete, stable, and valid;
   2. The software architecture is consistent:
      (a) Internally among the software architecture components, and
      (b) With the system architecture and operational concepts;
   3. The software architecture representations document the software architecture components and their interfaces, including:
      (a) Internal software-to-software and software-to-hardware interfaces;
      (b) External software-to-software and software-to-hardware interfaces; and
      (c) Data and control in each of these interfaces.
   4. The software architecture representations cover multiple perspectives, including:
      (a) Definition of logical architecture components, connectors, and interfaces, including their functionality, interactions, and dependencies;
      (b) Definition of the conceptual and logical data schema for key data structures and their relationship to the software architecture components and algorithms;
      (c) Definition of the software architecture component behaviors, interactions, and collaborations;
      Note: Techniques such as sequence diagrams, activity diagrams, and state machines can be used here.
      (d) Definition of the physical organization of the software, including the target processors on which the software architecture components execute; and
      (e) Identification and definition of the software units and a mapping between the software units and the software architecture components;
   5. Analysis of software architecture and design decisions and tradeoffs is valid, appropriate to the project scope and complexity, and documented;
   6. Evaluation of reusable software products, including commercial-off-the-shelf (COTS) software, supports their selection for partially or fully implementing one or more software architecture components;
   7. The software architecture is able to support growth and change and adequately isolates implementation details to facilitate change; and
   8. The software architecture has been evaluated as capable of meeting its allocated software requirements (e.g., by engineering analyses, modeling, and simulation).

c. The software qualification test planning is sufficiently mature to proceed with dependent software and system development activities, including the following aspects:

   1. The software qualification test plan is complete, stable, and valid;
(2) The software qualification test plan is consistent with the higher-level test plans;
(3) The software qualification test plan is consistent with the qualification provisions for verification methods and levels in the Software Requirements Specification(s) and Interface Requirements Specification(s);
Note: The qualification provisions are found in Section 4 of the SRS and IRS DIDs (See Section 6.2 for the SRS and IRS DIDs identifiers.)
(4) All software requirements, including interface and specialty engineering requirements, are:
   (a) Allocated to one or more tests in the Software Test Plan where they will be verified, and
   (b) Bidirectionally traced to and from the tests described in the Software Test Plan;
(5) The tests in the Software Test Plan include nominal, off-nominal, and error conditions; and
(6) The software requirements will be fully verified by the tests described in the Software Test Plan, whether the requirements are implemented by COTS software, other reusable software, including unmodified or modified reusable software, or newly developed software.
d. Implementation of the software integration and qualification test environment is sufficiently mature to proceed with software and system development activities that will use these facilities, including the following aspects:
   (1) The software integration and qualification test environment is in place, or planned to be in place, by the time needed for the build(s) under review;
   (2) The software integration and qualification test environment, in place or planned to be in place, has sufficient capability and capacity for the testing of the build(s) under review;
   (3) The software integration and qualification test environment, in place or planned to be in place, has all target hardware in a configuration as close to the operational configuration as possible; and
   (4) The software integration and qualification test environment has actual interfacing equipment and systems, or high-fidelity simulator(s) in place, or planned to be in place, by the time needed.

E.3.3 Software Build Design Review (SBDR)
This subsection provides the specific objectives for the Software Build Design Review (SBDR). All objectives apply to the software products as completed for the build(s) under review. See Section E.3 for overall requirements that apply to all software-specific joint technical reviews.
Note: The software requirements are specified in the Software Requirements Specification (SRS) and the Interface Requirements Specification (IRS). (See Section 6.2 for the SRS and IRS DIDs identifiers.)
1. The specific objectives of the SBDR shall be to ensure that:
   a. The software design, including the interface and database design, is adequate for meeting the software requirements, including interface requirements, specialty engineering requirements (e.g., Human Systems Integration, information assurance, dependability), performance requirements (Key Performance Parameters), response time requirements, and end-to-end timing requirements:
      (1) The software design has been demonstrated to meet all allocated software requirements;
      (2) The software design has been demonstrated to meet its allocated performance requirements with sufficient margin for this point in the lifecycle;
      (3) The software design has been demonstrated to meet its computer resource allocations for CPU, memory, storage, and I/O bandwidth with sufficient margin for this point in the lifecycle;
(4) All software requirements are allocated to software design components down to the software unit level; and

(5) All software requirements are bidirectionally traced to and from software design components down to the software unit level.

b. The software design, including the interface and database design, is consistent with the system and software architectures.

c. The software design, including the interface and database design, is sufficiently mature to proceed with the build’s dependent software and system development activities, including software implementation, integration, and test activities:

(1) The software design for the build has been elaborated to the level of software units, consistent with the SDP and the selected software lifecycle model(s);

(2) Software design decisions and their tradeoffs, including justifications for COTS and reusable software selections using evaluation criteria, were adequately analyzed and recorded;

(3) Each of the build’s software units is identified as containing COTS, unmodified reuse, modified reuse, or newly developed software, or a combination thereof.

(4) For each of the build’s software units containing COTS, unmodified reuse, or modified reuse software, the source of that software is identified.

(5) COTS products were evaluated and selected for implementing software architecture components and all tradeoffs, selections, installation, and configuration design decisions are recorded;

(6) Detailed software interface definitions are managed by Interface Design Documents (IDDs) or Interface Control Documents (ICDs) and agreed to by all parties; and

(7) The software, database, and interface design descriptions are clear, correct, complete, consistent, and unambiguous, and adequately address:

(a) Detailed design of all external and internal interfaces;

(b) Detailed design of all files, databases, shared memory, etc., and their storage and access methods;

(c) Detailed design of user interface screens and human system interactions if applicable;

(d) Detailed design of algorithms for the build’s software units, including mathematical and procedural algorithms;

(e) Detailed design of the dynamic structure of the build’s software (e.g., processes or tasks, flow of execution control, priorities, sequencing, dynamic creation and deletion of processes or tasks);

(f) Detailed design of glue code for integrating the COTS and other reusable software with the rest of the software;

(g) Detailed design of fault management and recovery methods, including safe mode for onboard software, exception handling, error handling, and operator notification of errors; and

(h) Application programming interfaces to be used.

d. The software detailed design is consistent with all applicable standards (e.g., interface, screen design, and open systems standards).

e. The software processes, including coding and testing standards, are sufficiently defined, mature, and effective for developing the software needed to meet the requirements and are suitable for the program scope and complexity, including:

(1) The selected programming language(s) to be used have been evaluated, justified, and recorded; and

(2) The coding standards to avoid vulnerabilities and facilitate maintenance are defined and recorded.
f. The software item *qualification test* plans and *cases* are sufficient to ensure thorough nominal and off-nominal testing of the software to demonstrate that the software requirements are verified in the target environment:

1. The software item *qualification test cases* in the Software Test Description (STD) are valid, complete, stable, and consistent with the software architecture and design and Software Test Plan (STP);

2. The software item *qualification test cases* are consistent with the qualification provisions for verification methods and levels specified in the Software Requirements Specification(s) and Interface Requirements Specification(s);

   Note: The qualification provisions are found in Section 4 of the SRS and IRS DIDs. (See Section 6.2 for the SRS and IRS DIDs identifiers.)

3. The software requirements, including interface requirements, are bidirectionally traced to and from *qualification test cases* described in the STD;

4. All software requirements, including interface requirements, are allocated to the test cases described in the STD where they will be verified;

5. The software requirements, including interface requirements, will be fully verified by the test cases described in the STD, whether they are implemented by COTS software, other reusable modified or unmodified software, or newly developed software; and

6. The test cases in the STD include verification of all software requirements, including interface requirements, under nominal, off-nominal, and error conditions.

g. The *software engineering environment* and *software integration and qualification test environment* are established and have adequate capability and capacity to meet the software development, integration, and verification requirements and schedules, including multiple concurrent *builds*:

1. Integration and *qualification test* facilities, including target hardware in a configuration as close to the operational configuration as possible, and actual interfacing equipment and systems, or high-fidelity simulators, for development, integration, and qualification testing are in place, or planned to be in place in time.

h. The software requirements, architecture, design, *qualification test plans* and *test cases*, and the software master build plan are correct, consistent, complete, traceable, contain no TBXs, and are supported by engineering analyses.

**E.3.4 Software Build Test Readiness Review (SBTRR)**

This subsection provides the specific objectives for the Software Build Test Readiness Review (SBTRR). All objectives apply to the software products as completed for the build(s) under review. See Section E.3 for overall requirements that apply to all software-specific joint technical reviews.

1. The objectives of the SBTRR shall be to ensure that:

   a. The software under test is sufficiently mature to begin the formal *qualification test* event:

      1. The software under test is under configuration control by the software configuration management organization;

      2. The software under test is clearly defined in the Software Version Description (SVD);

      3. All known problems with the software are documented in discrepancy and change reports (DCRs);

      4. No severity 1 or 2 DCRs are open for the software under test; and

      5. The impacts of any open DCRs on the test execution are known, and workarounds for affected procedures are documented and in place.

   b. The software item *qualification test* plans, *test cases*, *test procedures*, *test data*, and *test databases* are under configuration control by the software configuration management organization.

   c. The software item *qualification test* plans, *cases*, and *procedures* are correct, consistent, complete, and traceable to the software and interface requirements:
(1) The software item test plan, test cases, and test procedures in the STP and STD, including all changes to the software requirements, test plan, and test cases since the SBDR, are correct, consistent, complete, and traceable to the software and interface requirements (SRS, IRS).

Note: The software requirements are specified in the Software Requirements Specification (SRS) and the Interface Requirements Specification (IRS). (See Section 6.2 for the SRS and IRS DID identifiers.)

(2) The test procedures:
   (a) Include testing of both nominal and off-nominal conditions;
   (b) Include procedures for test setup, test execution, data capture, and data analysis;
   (c) Include bidirectional traceability between the software and interface requirements and the test procedure step where the requirements are verified;
   (d) Are sufficient to verify all of the requirements allocated to the test case;
   (e) Are consistent with the verification methods and levels in the SRS and IRS for each requirement;
   (f) Are sufficiently detailed to be repeatable; and
   (g) Are automated to the extent feasible.

(3) No changes to the software architecture or design since the SBDR have affected the test cases and test procedures for this test event.

d. Test management processes and procedures are in place to ensure that:
   (1) The processes to be followed during the test execution are defined and documented and will result in a controlled and disciplined test execution, including:
       (a) The processes for initial testing, retesting, and regression testing; and
       (b) The process for discrepancy adjudication to determine whether and how testing can be continued after a discrepancy has occurred during execution.
   (2) Personnel roles and responsibilities are well defined for developer, acquirer, and acquirer support personnel.
   (3) Sufficient time on the software integration and qualification test environment is available for the qualification test event to execute all test procedures without schedule compression, including sufficient schedule margin for retesting.
   (4) The test schedule is feasible for the test execution to be performed.
   (5) Software quality assurance personnel are present to ensure that:
       (a) The test processes are followed;
       (b) The test procedures are executed;
       (c) Any deviations from the test procedures or test results are documented as redlines;
       (d) All problems and discrepancies are documented in DCRs; and
       (e) The test log completely and accurately documents the test execution, including test start, test end, interruptions, and discrepancies.

Note: See Appendix F for test log requirements.

e. The software, test procedures, and test data have been successfully dry run:
   (1) The software item qualification test procedures include both nominal and off-nominal conditions, and both nominal and off-nominal conditions have been successfully dry run with the expected results;
   (2) The software item qualification test procedures have been successfully dry run in the software integration and qualification test environment using established test data:
       (a) The test procedures have been successfully dry run, and the redlines from the dry run have been incorporated into the test procedures; and
       (b) Any problems or discrepancies encountered during the procedure dry runs have been captured in DCRs.
(3) The software item qualification test procedures were able to verify their allocated software and interface requirements with their expected results in the software integration and qualification test environment; and

(4) Disciplined test processes are in place, and all necessary test resources, including test personnel and the software integration and qualification test environment, are available.

f. The software integration and qualification test environment is ready for the formal qualification test event to begin:

(1) The software integration and qualification test environment is clearly defined and is under configuration control by the software configuration management organization, including:
   (a) both the hardware and software, including test tools and high-fidelity simulators, and
   (b) all test data and test databases;

(2) The software integration and qualification test environment, including the test data and databases, is sufficient to adequately verify the software and interface requirements;

(3) The software integration and qualification test environment has undergone sufficient validation to ensure all items in the test environment, including hardware, software, and high-fidelity simulator(s), correctly perform the functions necessary to support the qualification test event; and

(4) The software integration and qualification test environment is fully representative of the operational environment including:
   (a) Operational hardware and software configurations, operational data rates, and operational scenarios;
   (b) Hardware-in-the-loop testing for software and system testing; and
   (c) Software in the loop for hardware and system testing.

E.3.5 Software Build Exit Review (SBER)

This subsection provides the specific objectives for the Software Build Exit Review (SBER). All objectives apply to the software products as completed for the build(s) under review. See Section E.3 for overall requirements that apply to all software-specific joint technical reviews.

1. The specific objectives of the SBER shall be to ensure that as completed for this build:
   a. The updated software requirements, architecture, and design, remain valid, complete, consistent, stable, and traceable, including:
      (1) Bidirectional traceability is maintained between the software and interface requirements in the SRS and IRS and the:
          (a) Software architecture and design components, and
          (b) Software item qualification test cases and test procedure steps.
   b. The software unit integration and regression test plans, cases, and procedures are correct, complete, consistent, stable, traceable, repeatable, address both nominal and off-nominal conditions, and were successfully executed for the build under review:
      (1) Build integration and build regression test results were documented in software development files (SDFs), and
      (2) Build regression testing has ensured that the build under review had no adverse effects on functions successfully developed and tested in previous build(s).
   c. The software item qualification test plan, cases, and procedures are correct, complete, consistent, stable, traceable, repeatable, address both nominal and off-nominal conditions, and were executed successfully, including:
      (1) Software item qualification test results were documented in the Software Test Report (STR), including:
(a) Test logs,
(b) As-run redlined test procedures corrected to be as run,
(c) Results of data capture and analysis,
(d) Problems encountered,
(e) DCRs opened,
(f) regression testing, and
(g) retesting.

Note: Retesting and regression testing are recorded in the test log portion of the STR.

d. All problems have been documented in DCRs and categorized for severity:
   (1) All severity 1 and 2 problems have been resolved,
   (2) Retests have been successfully executed,
   (3) The results have been documented, and
   (4) All open severity 3 problems have been dispositioned and assigned to future builds for resolution.

e. The software requirements verification status, i.e., fully verified, partially verified, not verified:
   (1) is being maintained for each SRS and IRS requirement, and
   (2) is in an up-to-date condition following software item qualification testing.

f. All SRS and IRS requirements that were scheduled for verification in this build but were not verified have been assigned to a future build for completion.

g. The software is sufficiently tested to proceed with dependent software, hardware, and system integration and test activities.

h. The software risks for the software in this build have been updated to include identification and assessment of new risks and reassessment of existing risks as a result of the completion of this build:
   (1) Updated software risk mitigation plans are in place, and
   (2) Software risk mitigation is proceeding according to the updated software risk mitigation plans.

i. Software size, effort, cost, schedule, and staffing estimates and actuals were updated following this build and are consistent with the build results.

j. The Software Master Build Plan (SMBP), including the build schedule, remains executable for future builds, considering:
   (1) The increased knowledge of the software requirements, architecture, design, implementation, test plans, test cases, test procedures, and test results;
   (2) The identified software risks;
   (3) The updated software size, effort, cost, schedule, and staffing estimates remain consistent with the build results; and
   (4) The adequacy of any updates to the SMBP since the SBDR and SBTRR, such as new DCRs allocated to the future builds or software requirements not satisfied by this and previous builds slipping to future builds.

k. The SDP and software standards and procedures (e.g., work instructions) are still adequate for the software development activities to be performed in future builds and for the program scope and complexity, including any changes made since the SBDR and SBTRR.

l. If this build is scheduled for delivery to the operational environment, the following are satisfied:
   (1) Adequate training has been performed for operations personnel;
   (2) The software is sufficiently mature for operations (e.g., the number of workarounds to handle open severity 3 DCRs is small enough that the operators can still perform their tasks within their required response times);
   (3) The software is ready for installation at user sites;
   (4) The user and operator manuals are complete for this build;
The software version descriptions are complete for this build; and
The software installation preparations, activities, and instructions are complete for this build.

If this build is scheduled for delivery to the maintenance environment, the following are satisfied:

1. The sustainment products are correct and complete:
   a. The software product specifications, and
   b. The software version descriptions;
2. The software transition to maintenance preparations, activities, and instructions are complete for this build;
3. The software is sufficiently mature for simultaneous operations and maintenance:
   a. e.g., the number of workarounds to handle open severity 3 DCRs is small enough that the operators can still perform their tasks within their required response times; and
   b. e.g., the number of maintenance and development personnel can support both maintenance and support anomalies or user problems in operations;
4. The resources for sustainment are adequate and in place;
5. The training has been adequate for sustainment personnel;
6. The transition of the software engineering environment, if applicable, is complete; and
7. The maintenance organization is ready to begin maintenance.

Based on all of the above criteria, the software is sufficiently mature to consider this build completed.

E.4 Joint Technical Reviews Supporting Major Reviews

Note: Major reviews include system-level reviews such as System Requirements Review (SRR), System Design Review (SDR), System Functional Review (SFR), Software Requirements and Architecture Review (SAR), Preliminary Design Review (PDR), Critical Design Review (CDR), and Test Readiness Review (TRR). One set of requirements for major reviews can be found in (Peresztegy 2009-1) and (Peresztegy 2009-2).

1. If a major review is required by the contract and its scope includes software, then a software joint technical review shall be held to assess the technical readiness of the software products and other materials prior to the major review.
2. The software joint technical review shall be held sufficiently in advance of the major review so that identified problems and issues can be resolved before the delivery of products and other materials in support of the major review.

E.5 Joint Management Reviews

The following requirements supplement those in Section 5.18.2 for joint management reviews.

E.5.1 Joint Management Review Frequency

1. The developer shall hold periodic joint management reviews:
   a. At a frequency agreed to by the acquirer and developer or as specified by the contract, and
   b. At least quarterly.
2. The developer shall hold joint management reviews on an event-driven basis:
   a. Prior to software-related project milestones as specified by the contract or as agreed to by the acquirer and developer;
   b. Following the joint technical review that precedes each major review, or in combination with that joint technical review (see Section E.4), and
   c. For software issues that need to be addressed between the periodic joint management reviews, as agreed to by the acquirer and developer.
E.5.2 Joint Management Review Topics

1. The software management products in Table E.5-1 shall be reviewed at each joint management review:
   a. Upon initial definition of the product, and
   b. For any changes or updates to the product.

2. The joint management reviews shall also address the following items:
   a. Open technical issues in the software products and activities, and
   b. Technical risks that are new or changed.

Table E.5-1 Software Management Products

<table>
<thead>
<tr>
<th>ID</th>
<th>Software Management Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Software-related plans</td>
</tr>
<tr>
<td>2</td>
<td>Software estimates</td>
</tr>
<tr>
<td>3</td>
<td>Integrated schedules</td>
</tr>
<tr>
<td>4</td>
<td>Analysis of impacts from entities external to the build(s) or software item(s) under review</td>
</tr>
<tr>
<td>5</td>
<td>Identified software-related risks and risk handling status</td>
</tr>
<tr>
<td>6</td>
<td>Identified discrepancy and change reports (DCRs) and status</td>
</tr>
<tr>
<td>7</td>
<td>Open software-related issues and status</td>
</tr>
<tr>
<td>8</td>
<td>Action items status for action items related to software</td>
</tr>
<tr>
<td>9</td>
<td>Software requirements verification status</td>
</tr>
<tr>
<td>10</td>
<td>Software-related product status</td>
</tr>
<tr>
<td>11</td>
<td>Software-related process execution status</td>
</tr>
<tr>
<td>12</td>
<td>Current software measurement reports, including interpretations of the latest measurement values and trends</td>
</tr>
<tr>
<td>13</td>
<td>Software configuration management status</td>
</tr>
<tr>
<td>14</td>
<td>Software quality assurance status, including noncompliance reports and status</td>
</tr>
<tr>
<td>15</td>
<td>Status of procurement, development, and maintenance of the software-related management environment, software engineering environment, and software integration and qualification test environment.</td>
</tr>
<tr>
<td>16</td>
<td>Analysis of impacts on entities external to the build under review</td>
</tr>
</tbody>
</table>
Appendix F. Test Log Requirements

F.1 Scope
This appendix is a MANDATORY part of this standard. This appendix identifies the minimum information that is required for test logs.

F.2 Test Logs
F.2.1 Test Log Definitions
1. If the developer has alternative definitions for the test log terms defined in Section F.2.1, then the developer shall provide the definitions of those terms in the Software Development Plan (SDP).

The test log terms that apply to integration testing and qualification testing are defined below in alphabetical order.

Note: The developer may record additional information in test logs at any level of integration and qualification testing.

Description of test incident. A description of how the test deviated from the defined test procedure or expected results for example, inconsistencies between the documented expected results compared to the actual results; an error in the test procedure or test script; missing or failed test equipment or simulators; incorrect configuration of test equipment or simulators.

Description of test interruption. A description of why or how the test activity or test procedure was interrupted. See Section 3.1 definition of test interruption for examples.

List of discrepancy and change report identifiers. References to the identifier(s) of the discrepancy and change report(s) (DCRs) that were written as a result of the discrepancy or test incident on:
1. the software under test,
2. the test cases and test procedures, or
3. the test environment.

Note: These DCRs usually have mechanisms to provide additional information or attachments that can help identify the cause of the discrepancy or test incident.

List of witnesses. The names and organizations of the witnesses present for the test procedure.

Start time of the test interruption. Wall clock time, including the date and time zone of the start of the particular test interruption.

Start time of the test procedure(s). Wall clock time, including the date and time zone of the start of the particular test procedure, or set of test procedures if they are automated.

Stop time of the test interruption. Wall clock time, including the date and time zone of the end of the particular test interruption.

Stop time of the test procedure(s). Wall clock time, including the date and time zone of the end of the particular test procedure, or set of test procedures if they are automated.

Test procedure name or identifier. The name or identifier of the test procedure, and if applicable, the step in the test procedure.
F.2.2 Test Log Requirements

1. The developer shall produce a test log for each of the following activities:
   a. Unit integration and testing (Section 5.8),
   b. Software item qualification testing (Section 5.9),
   c. Software-hardware item integration and testing (Section 5.10), and
   d. System qualification testing (Section 5.11).

2. The developer shall include in the test log, as a minimum, the following information for each testing activity listed above:
   a. Test procedure name or identifier;
   b. Start time of the test procedure(s);
   c. Stop time of the test procedure(s);
   d. Description of test interruption, if any;
   e. Start time of the test interruption, if any;
   f. Stop time of the test interruption, if any;
   g. Description of test incident, if any;
   h. Start time of the test incident, if any;
   i. Stop time of the test incident, if any;
   j. List of discrepancy and change report identifier(s), if any; and
   k. List of witnesses (only required for qualification testing).

3. The developer shall include in the test log, explicitly or by reference to configuration controlled documentation, a description of the test environment that includes, as a minimum:
   a. Identification of all hardware by manufacturer, model, type, serial number, detailed configuration, and, if applicable, calibration date(s);
   b. Identification of all operating systems in use by name, version, and release;
   c. Identification of all software in use by name, version, and release;
   d. Identification of all simulators and emulators by name, version, and release;
   e. Identification of all databases used by name, version, and release;
   f. Identification of all specialized test equipment by name, model, and detailed configuration; and
   g. Identification of all other pertinent environmental conditions required to perform the testing.
Appendix G. Reserved
Appendix H. Product Templates

Scope
This appendix is a MANDATORY part of this standard. This appendix provides templates for several documents. See Section 6.2 for DID identifiers to be used with these templates.

The content of each template is consistent with the requirements of this standard. The templates included in this appendix are as follows:

H.1 Software Development Plan (SDP)
H.2 Software Architecture Description (SAD)
H.3 Software Master Build Plan (SMBP)
H.4 Software Measurement Plan (SMP)
H.5 Software Measurement Report (SMR)
H.6 Process Improvement Plan (PIP)
H.1 Software Development Plan (SDP) Template

This appendix is a MANDATORY part of the (SDSMCS). It provides the content requirements for the SDP.

1. Referenced information cited in paragraphs 4, 5, 6, and 7 shall be provided as attachments to the plan.

2. If the section numbering used below is not used, the developer shall provide an appendix in the SDP with a traceability matrix mapping from the section numbers and titles below to the section numbers and titles used in the developer’s SDP.

3. If there is such a traceability mapping appendix, it shall be referenced in section 1.3.

Note 1: The numbering shown in paragraphs 4 and 5 below is consistent with the Software Development Standard for Mission Critical Systems (SDSMCS).

Purpose. The Software Development Plan (SDP) describes a developer’s plans for conducting a software development effort. The term “software development” is meant to include the new development, modification, reuse, reengineering, incorporation of commercial item (also known as COTS) packages, maintenance, and all other activities resulting in software products. The SDP provides the acquirer insight into, and a tool for monitoring, the processes to be followed for software development; the methods to be used; the approach to be followed for each activity; and project schedules, organization, and resources.

References


Content Requirements
This template contains the required content of the Software Development Plan (SDP). See Section 3 of the Software Development Standard for Mission Critical Systems (SDSMCS), Aerospace Report No. TOR-2013-00083, for definitions of all italicized words or phrases.

1. **Scope.** This section shall be divided into the following paragraphs.

   1.1 **Identification.** This paragraph shall contain a full identification of the system and the software to which this document applies, including, as applicable, identification number(s), title(s), abbreviation(s), version number(s), and release number(s).

   1.2 **System overview.** This paragraph shall briefly state the purpose of the system and the software to which this document applies. It shall: a) describe the general nature of the system and software; b)
summarize the history of system development, operation, and maintenance; c) identify the project sponsor, acquirer, user, developer, and support organizations; and d) identify current and planned operating and user sites.

1.3 Document overview. This paragraph shall summarize the purpose and contents of this document. This paragraph shall describe any security or privacy considerations associated with its use.

1.4 Relationship to other plans. This paragraph shall describe the relationship, if any, of the SDP to other project management plans.

2. Referenced documents. This section shall list the number, title, revision, and date of all documents referenced in this plan. This section shall also identify the source for all documents not available through normal Government stocking activities.

3. Overview of required work. This section shall be divided into paragraphs as needed to establish the context for the planning described in later sections. It shall include, as applicable, an overview of:
   a. Requirements and constraints on the system and software to be developed,
   b. Requirements and constraints on project documentation;
   c. Position of the project in the system lifecycle;
   d. The selected project and acquisition strategy;
   e. Any requirements or constraints on the selected project and acquisition strategy;
   f. Requirements and constraints on project schedules and resources; and
   g. Other requirements and constraints, such as on project security, privacy, methods, standards, interdependencies on hardware and software development.

4. General requirements. Provisions corresponding to nonrequired activities may be satisfied by the words “Not applicable.” If different builds or different software on the project require different planning, these differences shall be noted in the paragraphs. See Section 4 in the body of the Software Development Standard for Mission Critical Systems (SDSMCS) for the activities and topics to be addressed in this leading paragraph. This section shall be divided into the following paragraphs. In addition to the content specified below, each paragraph shall identify applicable risks and uncertainties and plans for dealing with them.

4.1 Software development process. This paragraph shall describe the software development process to be used. The planning shall cover: a) identification of the software development lifecycle model(s) to be used; b) planned builds, if applicable; c) their build objectives; and d) the software development activities to be performed in each build. See Section 4.1 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph.

4.2 General requirements for software development. See Section 4.2 and its subsections in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software development. This paragraph shall be divided into the following subparagraphs.

4.2.1 Software development methods. This paragraph shall describe or reference the software development methods to be used. This paragraph shall include descriptions of the manual and automated tools and procedures to be used in support of these methods. Reference may be made to other paragraphs in this plan if the methods are better described in context with the activities to which they will be applied. See Section 4.2.1 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software development methods.
4.2.2 Standards for products. This paragraph shall describe or reference the standards to be followed for representing requirements, architecture, design, code, test cases, test procedures, test results, test logs, and discrepancy and change reports. The standards shall cover all contractual requirements concerning standards for products. Reference may be made to other paragraphs in this plan if the standards are better described in context with the activities to which they will be applied. See Section 4.2.2 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on standards for software products.

The contents of Paragraph 4.2.2 shall be placed into separate appendices of the SDP, not in Paragraph 4.2.2. Paragraph 4.2.2 shall reference these appendices.

4.2.2.1 Standards for code. Standards for code shall be provided for each programming language to be used. The coding standards for each language shall include, as a minimum:

a. Standards for format (such as indentation, spacing, capitalization, and order of information);

b. Standards for header comments, requiring, for example, name and identifier of the code; version identification; modification history; purpose; requirements and design decisions implemented; notes on the processing (such as algorithms used, assumptions, constraints, limitations, and side effects); and notes on the data (e.g., inputs, outputs, variables, data structures);

c. Standards for other comments, such as required number and content expectations);

d. Naming conventions (e.g., for constants, types, variables, parameters, packages, procedures, classes, objects, methods, functions, files);

e. Restrictions, if any, on the use of programming language constructs or features; and

f. Restrictions, if any, on the complexity of code aggregates.

4.2.2.2 Standards for DCRs. Standards for discrepancy and change reports (DCRs) shall be provided. The DCR standards shall include, as a minimum:

a. A glossary and definitions of terms that can be used in discrepancy and change reports (DCRs), including all specialized terms used in DCR titles, descriptions, causes, and resolutions;

b. Alternative and additional definitions, if any, for DCR terms specified in Appendix C.2.1 of this standard;

c. A DCR acronym list that includes all acronyms that are used (or are permitted to be used) in DCRs. These acronyms might appear in (e.g., DCR titles, free text descriptions of test incidents, discrepancies, failures, causes, resolutions, and development, integration and qualification test activity names);

Note: This DCR acronym list is in addition to the acronym list for the entire SDP.

d. A list of activity names and their definitions used for DCRs besides those in Appendix C, Table C.2-2, of the standard; and

e. The names and sequence of the DCR steps that can be used.

4.2.2.3 Standards for test logs. Standards for test logs shall be provided. The test log standards shall include, as a minimum:

a. The test log fields and terms specified in Appendix F.2 of (SDSMCS); and

b. Alternative and additional definitions, if any, for test log terms specified in Appendix F.2.1 of (SDSMCS).
4.2.3 Traceability. This paragraph shall describe the approach to be followed for establishing and maintaining bidirectional traceability between levels of requirements, between requirements and design, between design and the software that implements it, between requirements and qualification test information, and between computer hardware resource utilization requirements and measured computer hardware resource utilization. See Section 4.2.3 in the body of (SDSMCS) for the activities, topics, and other items to be addressed in this paragraph on bidirectional traceability.

4.2.4 Reusable software products. See Section 4.2.4 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on reusable software products. This paragraph shall be divided into the following subparagraphs.

4.2.4.1 Incorporating reusable software products. This paragraph shall describe the approach to be followed for identifying, evaluating, and incorporating reusable software products, including the scope of the search for such products and the criteria to be used for their evaluation. Candidate or selected reusable software products known at the time this plan is prepared or updated shall be identified and described, together with benefits, drawbacks, alternatives considered, rationale for those selected, remaining viable alternatives, and restrictions, as applicable, associated with their use.

4.2.4.2 Developing reusable software products. This paragraph shall describe the approach to be followed for identifying, evaluating, and reporting opportunities for developing reusable software products.

4.2.5 Assurance of critical requirements. See Section 4.2.5 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on assurance of critical requirements. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for handling requirements designated critical.

4.2.5.1 Safety
4.2.5.2 Security
4.2.5.3 Privacy protection
4.2.5.4 Reliability, maintainability, and availability
4.2.5.5 Dependability
4.2.5.6 Human system integration, including human factors engineering and
4.2.5.7 Assurance of other mission-critical requirements as agreed to by the acquirer and developer

4.2.6 Computer hardware resource utilization. This paragraph shall describe the approach to be followed for allocating computer hardware resources and monitoring their utilization. See Section 4.2.6 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on computer hardware resource utilization.

4.2.7 Recording rationale. This paragraph shall describe the approach to be followed for recording rationale that will be useful to the support organization for key decisions made on the project. It shall interpret the term “key decisions” for the project. It shall state where the rationale are to be recorded. See Section 4.2.7 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on recording rationale.

4.2.8 Access for acquirer review. This paragraph shall describe the approach to be followed for providing the acquirer and its authorized representatives access to developer and software team member facilities for review of products and activities. It shall cover all contractual requirements concerning acquirer team access for review. See Section 4.2.8 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on access for acquirer review.
4.2.9 **Contractual requirements.** This paragraph shall describe the approach to be followed for meeting all the contractual requirements regarding software development, including testing, transition, maintenance, and operations. Reference may be made to other paragraphs in this plan if the approach to be followed for meeting contractual requirements is better described in context with the activities to which they will be applied. These contractual requirements can be found in, e.g., the Statement of Work (SOW), Contract Data Requirements List (CDRL), compliance documents and their tailoring, Integrated Master Plan (IMP), specifications, Section H of the Model Contract (Sections A-K of the RFP and Contract), and other contractual documentation.

5. **Plans for performing detailed software development activities.** The paragraphs below cover the plans for performing detailed software development activities. Provisions corresponding to nonrequired activities may be satisfied by the words “Not applicable.” If different builds or different software on the project require different planning, these differences shall be noted in the paragraphs. If different planning is required for new development, modification, reusable software products, reengineering, and maintenance, these differences shall be described in the paragraphs. The discussion of each activity shall include the approach, i.e., plans, processes, methods, procedures, tools, roles, and responsibilities, to be applied to: 1) the analysis or other technical tasks involved, 2) the recording of results, and 3) the preparation of associated deliverables, if applicable. For each activity, include a) entrance criteria, b) inputs, c) tasks to be accomplished, d) products to be produced, e) verifications to be used to ensure tasks are performed according to their defined processes and products meet their requirements, f) outputs, and g) exit criteria. The discussion shall also identify applicable risks and uncertainties and plans for dealing with them. Reference may be made to paragraph 4.2.1 if applicable methods are described there. This section shall be divided into the following paragraphs.

5.1 **Project planning and oversight.** See Section 5.1 and its subparagraphs in the body of the Software Development Standard for Mission Critical Systems (SDSMCS) for the activities and topics to be addressed in this paragraph on project planning and oversight. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for project planning and oversight.

- 5.1.1 Software development planning
- 5.1.2 Software integration and test planning
  - 5.1.2.1 Software integration planning
  - 5.1.2.2 Software item qualification test planning
- 5.1.3 System qualification test planning
- 5.1.4 Planning for software transition to operations
- 5.1.5 Planning for software transition to maintenance
- 5.1.6 Following and updating plans

5.2 **Establishing a software development environment.** The developer shall record the results of the software engineering environment adequacy analysis in the SDP. The developer shall record the results of the software integration and qualification test environment adequacy analysis in the SDP. See Section 5.2 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on establishing and maintaining software development environments. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for establishing, controlling, and maintaining a software development environment.

- 5.2.1 Software engineering environment
  - 5.2.1.1 Software engineering environment description
  - 5.2.1.2 Software engineering environment adequacy analysis reports
- 5.2.2 Software integration and qualification test environment
  - 5.2.2.1 Software integration and qualification test environment description
5.2.2.2 Software integration and qualification test environment adequacy analysis reports

5.2.3 Software development library

5.2.4 Software development files

5.2.5 Nondeliverable software

5.3 System requirements analysis. See Section 5.3 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on system requirements analysis. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for participating in system requirements analysis.

5.3.1 Analysis of user input

5.3.2 Operational concept

5.3.3 System requirements definition

5.4 System architecture and design. See Section 5.4 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on requirements for system architectural design. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for participating in system architectural design.

5.4.1 System-wide architectural design decisions

5.4.2 System architectural design

5.5 Software requirements analysis. This paragraph shall describe the approach to be followed for software requirements analysis. See Section 5.5 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software requirements analysis.

5.6 Software architecture and design. See Section 5.6 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software architecture and design. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software design.

5.6.1 Overall software architecture

5.6.2 Software item architecture

5.6.3 Software item detailed design

5.6.3.1 Software unit detailed design

5.6.3.2 Software interface design

5.6.3.3 Database design, as applicable

5.6.3.4 User interface design, as applicable

5.6.3.5 Other applicable software design (e.g., model-based software, as applicable)

5.7 Software implementation and unit testing. See Section 5.7 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software implementation and unit testing. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software implementation and unit testing.

5.7.1 Implementing software

5.7.2 Preparing for unit testing

5.7.3 Performing unit testing

5.7.4 Analyzing and recording unit testing results

5.7.5 Unit regression testing

5.7.6 Revising and retesting units

5.8 Unit integration and testing. See Section 5.8 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software unit integration and testing.

H.1-6
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for unit integration and testing.

5.8.1 Testing on the target computer system
5.8.2 Preparing for unit integration and testing
5.8.3 Performing unit integration and testing
5.8.4 Analyzing and recording unit integration and test results
5.8.5 Unit integration regression testing
5.8.6 Revising and retesting unit integration

5.9 Software item qualification testing. See Section 5.9 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software item qualification testing. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software item qualification testing.

5.9.1 Independence in software item qualification testing
5.9.2 Testing on the target computer system
5.9.3 Preparing for software item qualification testing
5.9.4 Dry run of software item qualification testing
5.9.5 Performing software item qualification testing
5.9.6 Analyzing and recording software item qualification test results
5.9.7 Software item qualification regression testing
5.9.8 Revising and retesting software items

5.10 Software-hardware item integration and testing. See Section 5.10 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software-hardware item integration and testing. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for participating in software-hardware item integration and testing.

5.10.1 Testing on the target computer system
5.10.2 Preparing for software-hardware item integration and testing
5.10.3 Performing software-hardware item integration and testing
5.10.4 Analyzing and recording software-hardware item integration and test results
5.10.5 Software-hardware item integration regression testing
5.10.6 Revising and retesting software-hardware item integration

5.11 System qualification testing. See Section 5.11 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on system qualification testing. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for participating in system qualification testing.

5.11.1 Independence in system qualification testing
5.11.2 Testing on the target computer system(s)
5.11.3 Preparing for system qualification testing
5.11.4 Dry run of system qualification testing
5.11.5 Performing system qualification testing
5.11.6 Analyzing and recording system qualification test results
5.11.7 System qualification regression testing
5.11.8 Revising and retesting the system

5.12 Preparing for software transition to operations. See Section 5.12 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on preparing for software transition to operations. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for preparing for software transition to operations.
5.12.1 Preparing the executable software
5.12.2 Preparing version descriptions for user sites
5.12.3 Preparing user manuals
   5.12.3.1 Software user manuals
   5.12.3.2 Computer operations manuals
5.12.4 Installation at user sites

5.13 Preparing for software transition to maintenance. See Section 5.13 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on preparing for software transition to maintenance. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for preparing for software transition to maintenance.

5.13.1 Preparing the executable software
5.13.2 Preparing source files
5.13.3 Preparing version descriptions for the maintenance site(s)
5.13.4 Preparing the “as built” software architecture, design, and related information
5.13.5 Updating the system/subsystem design description
5.13.6 Updating the software requirements
5.13.7 Updating the system requirements
5.13.8 Preparing maintenance manuals
   5.13.8.1 Computer programming manuals
   5.13.8.2 Firmware support manuals
5.13.9 Transition to the designated maintenance site

5.14 Software configuration management. See Section 5.14 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software configuration management. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software configuration management.

5.14.1 Configuration identification
5.14.2 Configuration control
5.14.3 Configuration status accounting
5.14.4 Configuration audits
5.14.5 Packaging, storage, handling, and delivery
5.14.6 Baselines

5.15 Software peer reviews and product evaluations. See Section 5.15 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software peer reviews and product evaluations. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software peer reviews and product evaluations.

5.15.1 Software peer reviews
   5.15.1.1 Plan for software peer reviews
   5.15.1.2 Prepare for an individual peer review
   5.15.1.3 Conduct peer reviews
   5.15.1.4 Analyze and report peer review data
5.15.2 Product evaluations
   5.15.2.1 In-process and final product evaluations
   5.15.2.2 Product evaluation records
   5.15.2.3 Independence in product evaluations

5.16 Software quality assurance. See Section 5.16 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software quality assurance. This
paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software quality assurance.

5.16.1 Software quality assurance evaluations
5.16.2 Software quality assurance records
5.16.3 Independence in software quality assurance
5.16.4 Software quality assurance noncompliance issues

5.17 Corrective action. See Section 5.17 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on corrective action. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for corrective action.

5.17.1 Discrepancy and change reports (DCRs)
These DCRs shall include the items to be recorded specified in Appendix C, Table C.2-5 of (SDSMCS).

5.17.2 Corrective action system

5.18 Joint technical and management reviews. See Section 5.18 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on joint technical and management reviews. See Appendix E, Joint Technical and Management Reviews for additional requirements for joint technical and management reviews. This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for joint technical and management reviews.

5.18.1 Joint technical reviews
5.18.2 Joint management reviews

5.19 Software risk management. This paragraph shall describe the approach for performing risk management. See Section 5.19 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software risk management.

5.20 Software measurement. This paragraph shall briefly summarize the approach to be used for software measurement throughout the system development lifecycle. This paragraph shall also itemize the specific software measurements to be collected, analyzed, interpreted, applied, and reported. In addition, this paragraph shall summarize the importance of each specific measurement used for decision making, corrective actions, and reporting to the acquirer. See Section 5.20 and its subparagraphs in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software measurement. When a separate software measurement plan (SMP) is not required on contract, this paragraph shall include the content described in the SMP template provided in Appendix H.4. When a separate SMP is required on contract, this paragraph shall include a reference to the SMP.

5.20.1 Software measurement planning
5.20.2 Software measurement reporting
5.20.3 Software measurement working group (SMWG)

5.21 Security and privacy. This paragraph shall describe the approach for meeting the security and privacy requirements. See Section 5.21 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on security and privacy.

5.22 Software team member management. This paragraph shall list all software developers at any level (e.g., prime contractor, software team members). This paragraph shall identify all software, including custom, COTS, modified, and reused, developed by foreign contractors at any level (e.g.,
prime contractor, software team members) that will be delivered to the acquirer. This paragraph shall identify the foreign contractor’s company name and foreign location(s). A “foreign contractor” means any foreign corporation, business association, partnership, trust, society or any other entity or group that is not incorporated or organized to do business in the United States, as well as international organizations, foreign Governments, and any agency or subdivision of foreign Governments (e.g., diplomatic missions). This paragraph shall describe the approach for performing software team member management. This paragraph shall specify the mechanisms to be used to ensure that all contractual requirements, and all changes to contractual requirements, are flowed down to all levels of software team members. See Section 5.22 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on software team member management.

5.23 Interface with software independent verification and validation (IV&V) agents. This paragraph shall describe the approach for interfacing with the software IV&V agents. See Section 5.23 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on interfacing with software independent verification and validation agents.

5.24 Coordination with associate developers. This paragraph shall describe the approach for performing the coordination with associate developers, working groups, and interface groups. See Section 5.24 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on coordination with associate developers.

5.25 Improvement of project processes. This paragraph shall describe the approach for performing the improvement of project processes. See Section 5.25 in the body of (SDSMCS) for the activities and topics to be addressed in this paragraph on improvement of project processes.

6. Schedules and activity network. This section shall be divided into the following paragraphs:

6.1 Schedule. This paragraph shall present schedule(s) identifying the activities and showing initiation of each activity, availability of draft and final deliverables, other milestones, and completion of each activity. This paragraph shall provide the detailed schedule activities for each software item, and other software, for each build, and for the entire software development lifecycle. See paragraph 7.2.1.4 below for inclusion of the rationale for the software cost and schedule estimation, including software cost and schedule estimation techniques, the input to those techniques (e.g., software size and software cost driver parameters and scale factors), and any assumptions made.

6.2 Activity network. An activity network depicting sequential relationships and dependencies among activities and identifying those activities that impose the greatest time restrictions on the project.

7. Project organization and resources. This section shall be divided into the following paragraphs to describe the project organization and resources.

7.1 Project organization. This paragraph shall be divided into the following subparagraphs to describe the organizational structure to be used on the project, including the organizations involved, their relationships to one another, and the authority and responsibility of each organization for carrying out required activities. Note: COTS software suppliers are not included in this paragraph and its subparagraphs.

7.1.1 Software team members. This paragraph shall identify each geographic site of each software team member organization that is performing the project-related effort for any software-related activities. (See Section 3.1 of the body of (SDSMCS) for the definition of software team member.)
For each software team member organization site this paragraph shall include all of the following information:

a. Organization site name (e.g., XYZ Co. Div ABA, City, State);
b. Parent organization name (e.g., company);
c. Internal organization name, i.e., name of division or other level (e.g., ground software development);
d. Organization site location, i.e., city, state, and country;
e. Software-related activities that the software team member is expected to be performing at this site; and
f. The internal structure of each software team member, showing all software-related entities (e.g., software development groups, test groups, software process organizations, software quality assurance, software configuration management) and how they relate to other project and organizational entities (e.g., program management, systems engineering, system integration and test, hardware engineering, quality assurance, configuration management).

Note: References to supplied organization charts could provide this information for item f.

7.1.2 Full set of project software. This paragraph shall identify the full set of software items and other software for all categories of software for this project. This paragraph shall include for each software item and other software:

a. Name of the software item or other software;
b. System, subsystem, and any other components to which the software belongs;
c. Category of software;
d. For each software team member responsible for all or part of the software item or other software, identify the:
   (1) Responsible software team member;
   (2) Organization site name, i.e., same as in 7.1.1.a;
   (3) Part(s) of the software item or other software for which the software team member is responsible;
   (4) Source of software, i.e., new, reused as is, modified reuse, COTS; and
   (5) If the software is of mixed source, then list the percentages of each type of source.

7.1.3 Software team receiver-giver relationships. This paragraph shall show the contractual and intracorporation receiver-giver relationships among the software team members, including the prime contractor. An internal software team member receiver is the software team member requiring and receiving a software item, or other software, from one of the other software team members. An internal software team member giver is the software team member supplying or giving the required software item or other software to the software team member that required the product. This paragraph shall be organized by internal software team member receiver with the prime contractor relationships first. If a single software team member has multiple sites performing software-related efforts on this project, then each organization site shall be treated separately. For each internal receiver-giver pair, this paragraph shall identify each software item, and other software, produced by the internal giver for the internal receiver. This paragraph shall include for each receiver-giver software team member relationship:

a. Internal software team member receiver organization site name,
b. Internal software team member giver organization site name, and
c. List of name(s) of each software item and other software to be produced by the giver for the receiver.
7.2 Project resources. This paragraph shall be divided into the following subparagraphs to describe the resources to be applied to the project.

7.2.1 Personnel resources. This paragraph shall provide the following items for the entire software development lifecycle and for each software item, and other software:

7.2.1.1 Staff hours by software item. The estimated staff-loading for the project i.e., number of total personnel hours by month throughout the system development lifecycle, broken out as follows:
   a. For each software team member:
      (1) For each software item and other software,
      (2) For each other piece of software,
      (3) For each build, and
      (4) For the entire software development lifecycle; and
   b. For all software team members for the entire software development effort.

7.2.1.2 Staff hours by responsibility. The breakdown of the staff-loading for the project, i.e., number of total personnel hours by month throughout the system development lifecycle, broken out by responsibility (for example, management, software engineering, software testing, software configuration management, product evaluation, software quality assurance):
   a. For each software team member:
      (1) For each software item,
      (2) For each other piece of software,
      (3) For each build, and
      (4) For the entire software development lifecycle; and
   b. For all software team members for the entire software development effort.

7.2.1.3 Number of personnel by skill level. For each software team member, a breakdown of the number of personnel by skill level of those personnel performing each responsibility used in paragraph 7.2.1.2:
   a. For each software team member:
      (1) For each software item,
      (2) For each other piece of software,
      (3) For each build, and
      (4) For the entire software development lifecycle; and
   b. For all software team members for the entire software development effort.

7.2.1.4 Rationale. The rationale for the schedule estimates in paragraph 6.1 and the effort and head count estimates in section 7.2, including software cost and schedule estimation techniques, the input to those techniques (e.g., software size and software cost driver parameters and scale factors), and any assumptions made.

7.2.1.5 Training. Description of the training required for each software team member organization site. Also a description of the training required for each new staff member.

7.2.2 Overview of developer facilities. For each organization site, this paragraph shall list the development and test facilities, secure areas, and other site features to be used, as applicable to the software development, including which work will be performed at each facility, area, or other site feature. This paragraph shall include a schedule detailing when these items will be needed, developed or acquired, and validated.

7.2.3 Acquirer-furnished equipment and information. This paragraph shall list acquirer-furnished equipment, software, services, documentation, data, and facilities, as applicable, required for the
software development effort. A schedule detailing when these items will be needed shall also be included.

7.2.4 Other required resources. This paragraph shall list other required resources, including a plan for obtaining the resources, dates needed, and availability of each resource item.

8. Notes. This section shall contain any general information that aids in understanding this document (e.g., background information, glossary, rationale). This section shall be divided into the following paragraphs.

8.1 Abbreviations and acronyms. This paragraph shall include an alphabetical listing of all acronyms, abbreviations, and their meanings as used in this document.

8.2 Glossary. This paragraph shall include a list of any terms and their definitions needed to understand this document. Terms often used differently between organizations (e.g., acquisition phase names, build, block, development phase names, effectivity, evolution, increment, and iteration) shall be defined to avoid confusion. If the terms used are exactly as defined in the Software Development Standard (SDSMCS), they need not be redefined here.

8.3 General information. This paragraph shall contain any other general information that aids in understanding this document (e.g., background information, rationale).

A. Appendices. Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling. Appendices shall be lettered alphabetically (Appendix A, B, etc.).

END of SDP Template
H.2 Software Architecture Description (SAD) Template

This appendix is a MANDATORY part of the (SDSMCS). It provides the content requirements for the Software Architecture Description (SAD).

1. If the section numbering used below is not used, the developer shall provide an appendix in the SAD with a traceability matrix mapping from the section numbers and titles below to the section numbers and titles used in the developer’s SAD.

2. If there is such a traceability mapping appendix, it shall be referenced in Section 1.3.

Purpose: The Software Architecture Description (SAD) documents the software architecture, including the approach, important design decisions, rationale, and tradeoffs. It provides diagrams and text to document the various architectural views from different perspectives and serves as the basis for the detailed design.

References

Content Requirements
This template contains the required content of the SAD. See Section 3 of the Software Development Standard for Mission Critical Systems (SDSMCS) for definitions of all italicized words or phrases.

1. Scope. This section shall be divided into the following paragraphs.

   1.1 Identification. This paragraph shall contain a full identification of the system and the software to which this document applies, including, as applicable, identification number(s), title(s), abbreviation(s), version number(s), and release number(s).

   1.2 System overview. This paragraph shall briefly state the purpose of the system and the software to which this document applies. It shall: a) describe the general nature of the system and software; b) summarize the history of system development, operation, and maintenance; c) identify the project sponsor, acquirer, user, developer, and support organizations; and d) identify current and planned operating and user sites.

   1.3 Document overview. This paragraph shall summarize the purpose and contents of this document. This paragraph shall describe any security or privacy considerations associated with its use.

   1.4 Relationship to other documents. This paragraph shall describe the relationship, if any, of the SAD to other project documents.

2. Referenced documents. This section shall list the number, title, revision, and date of all documents referenced in this plan. This section shall also identify the source for all documents not available through normal Government stocking activities.

3. Software architecture plans and processes. This section shall be divided into paragraphs as specified below to describe the plans and processes for developing the software architecture.
3.1 **Software architecture plans.** The paragraph shall provide an overview of the software architecture plans and activities. This paragraph shall identify the software team members responsible for developing and evaluating the software architecture, including their responsibilities. This paragraph shall also identify other stakeholders in the software architecture, including their roles.

3.2 **Software architecture processes and tools.** This paragraph shall describe how the software architecture has been developed in accordance with the detailed methods, techniques and tools specified in the Software Development Plan (SDP). This paragraph shall identify the architecture modeling tools and other tools and techniques (e.g., Unified Modeling Language) used to develop and maintain the software architecture, including representing, documenting, and analyzing the software architecture; performing consistency analyses; and mapping requirements to architectural components. This paragraph shall describe how the software architecture evolves from high-level architectural components and interfaces to lower level components and interfaces and how the lower-level components and interfaces will transition to the software design. This paragraph shall identify the scope of the software architecture; that is, what design decisions are and are not considered part of the software architecture.

4. **Software architecture requirements and approach.** This section shall be divided into paragraphs as specified below to describe the software architecture requirements and approach.

4.1 **Software architecture critical and driving requirements.** This paragraph shall identify: a) all critical requirements specified in the SDP (See (SDSMCS) Section 4.2.5); b) all other nonfunctional requirements that are drivers of software architecture decisions; and c) all functional requirements that are drivers of software architecture decisions. This paragraph shall contain a prioritized list of these critical and driving requirements according to their importance in software architecture decisions.

Note 1: *Nonfunctional requirements* are also called “quality attributes.” See definition of “nonfunctional requirement” in (SDSMCS), Section 3, Terms.

Note 2: Providing the critical and driving requirements identifiers n lieu of the actual requirement statements is acceptable.

4.2 **Software architecture approach.** This paragraph shall discuss the selected architectural approach, as well as alternatives that were considered to address the critical and driving requirements specified in paragraph 4.1 above. This paragraph shall describe the analyses and trade studies that were performed to evaluate the architectural alternatives for their ability to satisfy the critical and driving requirements. This paragraph shall describe how the results of these analyses and trade studies support the selected software architecture approach.

4.3 **Software architecture evaluations.** This paragraph shall describe all evaluations performed, or to be performed, of the selected software architecture, either by a developer team, an acquirer team, or a combined team. For evaluations that have been performed, this paragraph shall describe the results of these evaluations, with references to analysis details, especially with respect to the ability of the selected software architecture to meet the critical and driving requirements specified in paragraph 4.1 above. This paragraph shall describe any changes to software architecture decisions as a result of these evaluations.

4.4 **Software architecture risks.** This paragraph shall identify the software risks for the selected software architecture. This paragraph shall describe any analyses performed to evaluate these risks, the results of those analyses, with references to analysis details, and any mitigation actions taken or being undertaken for the risks.

H.2-2
5. **Overall software architecture description.** This paragraph shall provide the overall software architecture for all software on the contract, including all categories of software covered by the contract (see Section 1.2.5.6 of the (SDSMCS)). The overall software architecture description provided in this section shall be at the level of granularity that crosses software items. The following topics shall be addressed to describe the overall software architecture. The following topics may be discussed in any order chosen by the developer.
   a. A high-level description and diagrams of the software architecture.
   b. A description of how the software architecture integrates into the system and subsystem architectures and addresses the system operations concept and the primary threads that the system supports.
   c. A description of the relationship between the software architecture and any external systems.
   d. A description of how the software architecture addresses the critical and driving requirements (identified in paragraph 4.1 above) and their impact on the architecture.
   e. A description of the architecture style(s), applied design principles, key software architectural patterns, and constraints.
   f. A description, expressed in a set of use cases, or equivalent, of how the software will interact with the users and with other systems and subsystems to meet system and software requirements, including use cases, or equivalent, for nominal and off-nominal (e.g., alternative, error, and fault) scenarios.
   g. Descriptions of the following software architectural views, including both diagrams and detailed textual descriptions. All diagrams shall be accompanied by descriptions of the functionality and behavior provided by the components. This paragraph shall describe how the views address the concerns of the relevant stakeholders. This paragraph shall provide the criteria used to determine consistency among the architectural views. If additional views are used by the developer to describe the software architecture, those views shall be included here. The views shall include the following information, along with the rationale and the alternatives that were explored:
      (1) Descriptions, including diagrams and text, of logical architecture components, connectors, and interfaces, both internal and external. This paragraph shall include the functionality of each component and connector and the interactions and dependency relationships among components. This paragraph shall include the conceptual and logical data schema for key data structures, along with a description of the relationship between these data structures and the software architecture and algorithms;
      (2) Descriptions, including diagrams and text, of the architecture component behaviors, interactions, and collaborations required by each use case, or equivalent, using techniques such as sequence diagrams, activity diagrams, and state machine diagrams. This paragraph shall also include descriptions of states and modes and transitions among them, as applicable. Descriptions of important internal component behaviors shall also be included;
      (3) Descriptions, including diagrams and text, of the physical organization of the software. This paragraph shall include the target processors, both physical and virtual, on which components will execute, and their interconnections. This paragraph shall describe how software components, connectors, and other elements will be allocated to the target processors. This paragraph shall describe how and where system data are stored and accessed. This paragraph shall identify important software-to-hardware interfaces. This paragraph shall identify any special purpose hardware and any special target processor characteristics that have software impacts; and
      (4) Identification and descriptions of the software items and other software in the overall software architecture, including all categories of software. This paragraph shall also
include the mapping of the software items and other software
to the software architectural components.

h. Identification of commercial off-the-shelf (COTS) software products that will be used to
implement part or all of any software architecture components, including:

(1) Relationship of each COTS software product to the software architecture component(s) it
implements, what part(s) of the component(s) are implemented by each COTS software
product, and whether the full capabilities of the COTS software products are used;

(2) Discussion of alternative products evaluated, the evaluation criteria used, and how each
product met the evaluation criteria;

(3) Discussion of the data rights, including licensing, associated with each COTS software
product; and

(4) Discussion of how any mismatches between the COTS software product and the
architecture will be resolved.

i. Identification of reusable software products, i.e., noncommercial off-the-shelf, that will be
used to implement part or all of any software architecture component, including:

(1) Relationship of each reusable software product to the software architecture component(s) it
implements, what part(s) of the component(s) are implemented by each reusable software
product, and whether the full capabilities of the reusable software products are used;

(2) For each reusable software product, a description of what is being reused (e.g.,
requirements, design, algorithms, code, test cases), and the magnitude of expected
modifications to the reusable software component;

(3) Discussion of alternative products evaluated, the evaluation criteria used, and how each
product met the evaluation criteria;

(4) Discussion of the data rights associated with each reusable software product; and

(5) Discussion of how any mismatches between the reusable software product and the
architecture will be resolved.

j. Description of how and where the architecture supports Modular Open Software Approach
(MOSA) principles.

k. Description of how and where the architecture supports net-centricity principles, if
applicable.

l. Description of how and where the software architecture supports information assurance and
cyber-security requirements, including security assurance assessment and certification and
accreditation activities). Examples of supporting descriptions include:

(1) Principles that guide the security design of the software within the system (e.g., use of
defense-in-depth, modularity and isolation of security-critical functionality,
nonbypassability of security function chokepoints);

(2) Identification of system security policies (e.g., identification and authentication, access
control, confidentiality, integrity, data provenance, nonrepudiation, accountability); and
how they will be enforced by the software architecture;

(3) Identification of policy decision points and policy enforcement points within the
software architecture, including the technology and product choices for each;

(4) Identification of security domains, security modes (e.g., system high, dedicated), and
cross-domain solutions, including the types of data that they must handle;

(5) Detailed descriptions for aspects of the system security design that require a high level
of security assurance (e.g., key management design supporting National Security
Agency Type 1 encryption); and

(6) Detailed descriptions for aspects of the software architectural design that help support
cyber resilience, that is, the ability of a system to operate in the face of persistent cyber
attacks and still support mission success (e.g., redundancy of components, request

H.2-4
throttling, virtualization or partitioning of resources, deployment of security application tools).

m. Description of how and where the software architecture implements the supportability of the system, that is, the repair, scheduled maintenance, and preventive maintenance required to retain the system in, or restore the system to, a state in which it can perform its required functions, including the ability of personnel to install, configure, and monitor computer products, identify exceptions or faults, debug or isolate faults to root-cause analysis, and provide hardware or software maintenance in pursuit of solving a problem and restoring the product into service.

n. Description of how and where the software architecture supports system reliability, maintainability, availability (RMA), and safety, including the architectural decisions made to support RMA and safety, the fault management architecture, use of other architectural features to address RMA and safety (e.g., redundancy, automated failover, fault tolerance), and uniform exception handling and recovery methods.

o. If applicable, a description of how and where the software architecture supports the human systems interface to account for human capabilities and limitations in the operations, maintenance, and support of the system. This description shall include architecture decisions concerning user interface screen design and user interaction mechanisms for user input and output. This description shall include, if applicable:
   (1) How the software architecture isolates the user interface from the application logic;
   (2) Principal design decisions made to ensure usability by the human operator;
   (3) Principal design decisions made to ensure that the user interface is internally consistent across all software in the overall software architecture;
   (4) Principal design decisions made to ensure that the user interface is consistent with widely used application user interfaces;
   (5) Principal design decisions made to ensure the quantity and frequency of data presented to the operator, including alarms, warnings, and error messages, are able to be assimilated and responded to by the operator within the needed response time; and
   (6) Applicable human systems interface standards (e.g., graphical user interface (GUI) standards) and how those standards are used within the architecture.

p. Description of how the software architecture supports the selected software development lifecycle model(s) and the integration of software and hardware in each software build and system increment.

q. Discussion of other principal and architecture-wide design decisions that are not covered by the above items. Examples include the following:
   (1) Applicable standards (e.g., interface standards, open system standards) and how those standards are used within the architecture;
   (2) Application programming interfaces (APIs) to be used;
   (3) Algorithms to be used;
   (4) Communications mechanisms (e.g., publish and subscribe message passing, calling sequences, shared memory, sockets) to be used between software entities and under which circumstances each mechanism is to be used; and
   (5) Definitions of uniform data storage and access methods.

r. Requirements traceability. This paragraph shall provide bidirectional traceability:
   (1) Between the software architecture components and the software requirements and software interface requirements; and
   (2) Between the use cases, or equivalent, and the software requirements and software interface requirements.

6. Software item architecture description. This paragraph shall provide the software architecture for the individual software items on the contract, including all categories of software covered by the contract (see Section 1.2.5.7 of the (SDSMCS)). This paragraph shall be divided into

H.2-5
subparagraphs to describe the software architecture of each software item. The paragraphs containing an “x” in their numbers and an “x” in their names shall be repeated for each software item “x.”

6.x Software architecture description for software item x <Insert Name>. This paragraph shall describe the software architecture for software item x. This software item architecture description for software item x may be included in this paragraph or in a separate appendix that is referenced from this paragraph. The software item architecture description provided in this paragraph shall be at the level of granularity that includes all the software units in the software item. The following topics shall be addressed to describe the software item architecture. (The following topics may be discussed in any order chosen by the developer.)

a. A high-level description and diagrams of the software item architecture. This paragraph shall also include a description of how the software item architecture evolves from and is consistent with the overall software architecture described in paragraph 5 above.

b. A description of how the software item architecture integrates into the system and subsystem architectures and how the software item architecture addresses the system operations concept and the primary system threads that the software item supports.

c. A description of the relationship between the software item architecture and any external systems.

d. A description of how the software item architecture addresses the critical and driving requirements, i.e., identified in Paragraph 4.1 above, allocated to software item x and their impact on the software item architecture.

e. A description of the architecture style(s), applied design principles, key software architectural patterns, and constraints that apply to the software item architecture.

f. A description, expressed in a set of use cases, or equivalent, of how the software item will interact with the users and with other systems and subsystems to meet system and software requirements, including use cases, or equivalent, for nominal and off-nominal (e.g., alternative, error, and fault) scenarios.

g. Descriptions of the following software architectural views, including both diagrams and detailed textual descriptions. All diagrams shall be accompanied by descriptions of the functionality and behavior provided by the software item architecture components. This paragraph shall describe how the views address the concerns of the relevant stakeholders. This paragraph shall provide the criteria used to determine consistency among the software item architectural views. If additional views are used by the developer to describe the software item architecture, those views shall be included here. The views shall include the following information for the software item architecture, along with the rationale and the alternatives that were explored:

1. Descriptions, including diagrams and text, of logical software item architecture components, connectors, and interfaces, both internal and external. This paragraph shall include the functionality of each component and connector and the interactions and dependency relationships among components. This paragraph shall include the conceptual and logical data schema for key data structures, along with a description of the relationship between these data structures and the software item architecture and algorithms;

2. Descriptions, including diagrams and text, of the software item architecture component behaviors, interactions, and collaborations required by each use case, or equivalent, using techniques such as sequence diagrams, activity diagrams, and state machine diagrams. This paragraph shall also include descriptions of states and modes and transitions among them, as applicable. Descriptions of important internal component behaviors shall also be included;

3. Descriptions, including diagrams and text, of the physical organization of the software item. This paragraph shall include the physical and virtual target processors on which
software item architecture components will execute and their interconnections. This paragraph shall describe how software item architecture components, connectors, and other elements will be allocated to the target processors. This paragraph shall describe how and where system data created or used by the software item are stored and accessed. This paragraph shall identify important software-to-hardware interfaces. This paragraph shall identify any special-purpose hardware and any special target processor characteristics that have impacts on the software item; and

(4) Identification and descriptions of the software units in the software item architecture. This paragraph shall also include the mapping between the software units and the software item architectural components.

h. Identification of commercial off-the-shelf (COTS) software products that will be used to implement part or all of any software item architecture components, including:
   (1) Relationship of each COTS software product to the software item architecture component(s) it implements, what part(s) of the component(s) are implemented by each COTS software product, and whether the full capabilities of the COTS software products are used;
   (2) Discussion of alternative products evaluated, the evaluation criteria used, and how each product met the evaluation criteria;
   (3) Discussion of the data rights, including licensing, associated with each COTS software product; and
   (4) Discussion of how any mismatches between the COTS software product and the software item architecture will be resolved.

i. Identification of reusable software products, i.e., noncommercial off-the-shelf, that will be used to implement part or all of any software item architecture component, including:
   (1) Relationship of each reusable software product to the software item architecture component(s) it implements, what part(s) of the component(s) are implemented by each reusable software product, and whether the full capabilities of the reusable software products are used;
   (2) For each reusable software product, a description of what is being reused (e.g., requirements, design, algorithms, code, test cases), and the magnitude of expected modifications to the reusable software component;
   (3) Discussion of alternative products evaluated, the evaluation criteria used, and how each product met the evaluation criteria;
   (4) Discussion of the data rights associated with each reusable software product; and
   (5) Discussion of how any mismatches between the reusable software product and the software item architecture will be resolved.

j. Description of how and where the software item architecture supports Modular Open Software Approach (MOSA) principles.

k. Description of how and where the software item architecture supports net-centricity principles, if applicable.

l. Description of how and where the software item architecture supports information assurance and cyber-security requirements, including security assurance assessment and certification and accreditation activities. Examples of supporting descriptions include:
   (1) Principles that guide the security design of the software item within the system (e.g., use of defense-in-depth, modularity and isolation of security-critical functionality, nonbypassability of security function chokepoints);
   (2) Identification of system security policies (e.g., identification and authentication, access control, confidentiality, integrity, data provenance, nonrepudiation, accountability) and how they will be enforced by the software item architecture;
   (3) Identification of policy decision points and policy enforcement points within the software item architecture, including the technology and product choices for each;
Identification of security domains, security modes (e.g., system high, dedicated), and cross-domain solutions, including the types of data that they must handle;

Detailed descriptions for aspects of the system security design that require a high level of security assurance (e.g., key management design supporting National Security Agency Type I encryption); and

Detailed descriptions for aspects of the software item architectural design that help support cyber resilience, that is, the ability of a system to operate in the face of persistent cyber attacks and still support mission success (e.g., redundancy of components, request throttling, virtualization or partitioning of resources, deployment of security application tools).

m. Description of how and where the software item architecture implements the supportability of the system, that is, the repair, scheduled maintenance, and preventive maintenance required to retain the system in, or restore the system to, a state in which it can perform its required functions, including the ability of personnel to install, configure, and monitor computer products, identify exceptions or faults, debug or isolate faults to root-cause analysis, and provide hardware or software maintenance in pursuit of solving a problem and restoring the product into service.

n. Description of how and where the software item architecture supports system reliability, maintainability, and availability (RMA), and safety, including the software item architectural decisions made to support RMA and safety, the fault management architecture, use of other architectural features to address RMA and safety (e.g., redundancy, automated failover, fault tolerance), and uniform exception handling and recovery methods.

o. If applicable, a description of how and where the software item architecture supports the human systems interface to account for human capabilities and limitations in the operations, maintenance, and support of the system. This description shall include software item architecture decisions concerning user interface screen design and user interaction mechanisms for user input and output. This description shall include, if applicable:

1. How the software item architecture isolates the user interface from the application logic;
2. Principal software item design decisions made to ensure usability by the human operator;
3. Principal software item design decisions made to ensure that the user interface of the software item is internally consistent across the software item architecture;
4. Principal software item design decisions made to ensure that the user interface of the software item is consistent with widely used application user interfaces;
5. Principal software item design decisions made to ensure the quantity and frequency of data presented to the operator, including alarms, warnings, and error messages, is able to be assimilated and responded to by the operator within the needed response time; and
6. Applicable human systems interface standards (e.g., graphical user interface (GUI) standards) and how those standards are used within the software item architecture.

p. Description of how the software item architecture supports the selected software development lifecycle model(s) and the integration of software and hardware in each software build and system increment.

q. Discussion of other principal and software item architecture-wide design decisions that are not covered by the above items. Examples include the following:

1. Applicable standards (e.g., interface standards, open system standards) and how those standards are used within the software item architecture;
2. Application program interfaces (APIs) to be used within the software item;
3. Algorithms to be used within the software item;
4. Communications mechanisms (e.g., publish and subscribe message passing, calling sequences, shared memory, sockets) to be used between software entities within the software item and under which circumstances each mechanism is to be used; and
5. Definitions of uniform data storage and access methods within the software item.
(6) Requirements traceability. This paragraph shall provide bidirectional traceability:

(a) Between the software item architecture components and the software item requirements and software item interface requirements, and

(b) Between the use cases, or equivalent, and the software item requirements and software item interface requirements.

7. Notes. This section shall contain any general information that aids in understanding this document (e.g., background information, glossary, rationale). This section shall be divided into the following paragraphs.

7.1 Abbreviations and acronyms. This paragraph shall include an alphabetical listing of all acronyms, abbreviations, and their meanings as used in this document.

7.2 Glossary. This paragraph shall include a list of any terms and their definitions needed to understand this document. Terms often used differently between organizations (e.g., acquisition phase names, build, block, development phase names, effectivity, evolution, increment, and iteration) shall be defined to avoid confusion. If the terms used are exactly as defined in the Software Development Standard (SDSMCS), they need not be redefined here.

7.3 General information. This paragraph shall contain any other general information that aids in understanding this document (e.g., background information, rationale).

A. Appendices. Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling. Appendices shall be lettered alphabetically (Appendix A, B, etc.).

END of SAD Template
H.3 Software Master Build Plan (SMBP) Template

This appendix is a MANDATORY part of the (SDSMCS). It provides the content requirements for the SMBP.

1. If the section numbering used below is not used, the developer shall provide an appendix in the SMBP with a traceability matrix mapping from the section numbers and titles below to the section numbers and titles used in the developer SMBP.

2. If there is such a traceability mapping appendix, it shall be referenced in section 1.3.

**Purpose:** The Software Master Build Plan (SMBP) includes plans for integrating and verifying the software consistent with the software development lifecycle model(s). (The SMBP is sometimes known as the Master Software Integration and Verification Plan.)

**References**

**Content Requirements**
This template contains the required content of the SMBP. The content and level of detail of the SMBP is expected to evolve as more information is available and captured. See Section 3 of the Software Development Standard for Mission-Critical Systems (SDSMCS) for definitions of all italicized words or phrases.

1. **Scope**. This section shall be divided into the following paragraphs.

   1.1 **Identification**. This paragraph shall contain a full identification of the system and the software to which this document applies, including, as applicable, identification number(s), title(s), abbreviation(s), version number(s), and release number(s).

   1.2 **System overview**. This paragraph shall briefly state the purpose of the system and the software to which this document applies. It shall: a) describe the general nature of the system and software; b) summarize the history of system development, operation, and maintenance; c) identify the project sponsor, acquirer, user, developer, and support organizations; and d) identify current and planned operating and user sites.

   1.3 **Document overview**. This paragraph shall summarize the purpose and contents of this document. This paragraph shall describe any security or privacy considerations associated with its use.

   1.4 **Relationship to other documents**. This paragraph shall describe the relationship, if any, of the SMBP to the SDP and other project management plans and project documents.

2. **Referenced documents**. This section shall list the number, title, revision, and date of all documents referenced in this plan. This section shall also identify the source for all documents not available through normal Government stocking activities.

3. **Software master build plan**. This section shall provide the philosophy and rationale for the decisions about the contents and progression of the builds. This section shall provide multiple
perspectives of the builds, including descriptions of which requirements are in which builds, which software units and software items are integrated in which builds, and the integration levels and qualification testing levels of the various builds. Build content descriptions shall include newly developed software, reusable software, and the software used for verification. This paragraph shall be divided into the following subparagraphs. The paragraphs containing “.x” in their numbers and “x” in their names shall be repeated for each build “x.”

3.1 Planned builds. This paragraph shall provide the philosophy and rationale for determining the builds, based upon, e.g., required capability need dates by the acquirer, a regular integration schedule (e.g., quarterly or monthly), an achievable build development size determined by number of requirements, number of lines of source code, size of executable software, or some other factor. This paragraph shall provide the names of the planned builds. This paragraph shall provide the names of the planned builds in relationship to internal milestone reviews, e.g., build reviews and major reviews, e.g., SAR, PDR, CDR, Test Readiness Reviews (TRRs). This paragraph shall provide the driving objectives of each build.

3.2 Build requirements contents. This paragraph shall provide the philosophy and rationale for determining which requirements go into which builds. This paragraph shall summarize the set of software requirements that will be included in each build. Note: The SMBP can include requirements at higher levels than software, e.g., element, segment, subsystem, and system requirements.

3.2.x Build x <Insert Name> requirements. This paragraph shall provide the list of software requirements that will be included in build “x.” For each requirement, the requirement identifier, a short requirement description, the requirements text, and the verification event, as a minimum, shall be included. For each requirement, this paragraph shall indicate if the requirement will only be partially implemented in build “x.”
Note 1: A table can provide this information.
Note 2: This information may be placed in an appendix referenced from this section.
Note 3: If a software item is developed in multiple builds, its requirements might not be fully implemented and verified until the final build. The planning identifies the subset of each software item’s requirements to be implemented in each build.

3.3 Build integration levels. This paragraph shall provide the philosophy and rationale for determining which builds are promoted to a higher level of integration. This paragraph shall provide the hierarchy of integration and integration testing. This paragraph shall specify responsibilities for each integration level in the integration hierarchy. This paragraph shall summarize which levels of integration will occur for each build. This paragraph shall summarize which levels of integration testing will occur for each build.
Note 1: A table can provide this information.
Note 2: It is beneficial to list the hardware on which the software executes, whether it is COTS hardware or special hardware being developed.

3.3.x Build x <Insert Name> integration levels. This paragraph shall provide the list of integration level(s) that will be performed for build “x.” For example, the build will integrate software units or software items:
   a. into part of a software item,
   b. into a whole software item,
   c. into multiple software items,
   d. with the hardware items on which they execute,
   e. into the subsystem level,
   f. into the system level,
3.4 Build contents and integration order. This paragraph shall provide the philosophy and rationale for determining which units are integrated in which order. This paragraph shall specify the location of each integration activity. This paragraph shall summarize the software items and software units, including reusable software, that are allocated to each build.

3.4.x Build x <Insert Name> contents and integration order. For build “x,” this paragraph shall provide the subparagraphs below.

3.4.x.1 Build x <Insert Name> contents. This paragraph shall provide the set of software items and software units that will be included in build “x.”
Note 1: A table can provide this information.
Note 2: This information may be placed in an appendix referenced from this section.

3.4.x.2 Build x <Insert Name> integration order. This paragraph shall provide the intended order of integrating the units for build “x.”
Note 1: A table can provide this information.
Note 2: This information may be placed in an appendix referenced from this section.

3.5 Build qualification testing levels. This paragraph shall provide the philosophy and rationale for determining which builds will undergo which levels of qualification testing. This paragraph shall specify responsibilities for each qualification test level. This paragraph shall specify the location of each qualification test event. This paragraph shall summarize the qualification testing level(s) for each build.

3.5.x Build x <Insert Name> qualification testing levels. This paragraph shall provide the list of qualification testing levels that will be performed for build “x.” For example, the build will be qualification tested at the following level(s), if any: software item, software level (e.g., some or all software items), element level, segment level, subsystem level, or system level. Note: A table can provide this information. This paragraph shall provide the allocation of requirements to specific qualification testing events. This paragraph shall specify responsibilities for each qualification testing event. This paragraph shall specify the location of each qualification testing event for build “x.”

3.6 Build deliveries. This paragraph shall provide the philosophy and rationale for determining: a) which builds will be delivered to a higher level for internal integration and testing; b) which builds will be delivered to a higher level for internal qualification testing; c) which builds will be delivered to the acquirer for integrated operational test and evaluation; and d) which builds will be delivered to the acquirer for operations. This paragraph shall summarize which builds will be delivered for internal integration and test. This paragraph shall summarize which builds will be delivered for higher levels of internal integration, integration testing, and qualification testing. This paragraph shall summarize which builds will be delivered to the acquirer for integrated operational test and evaluation. This paragraph shall summarize which builds will be delivered to the acquirer for operations.

3.7 Build schedule. This paragraph shall summarize when each build will be started and completed. This paragraph shall summarize when each build will be delivered for each level of integration. This paragraph shall summarize when each build will be delivered for each level of qualification testing. This paragraph shall summarize when each build will be delivered to the

H.3-3
acquirer for integrated operational test and operations. This paragraph shall summarize when each build will be delivered to the acquirer for operations.

4. Notes. This section shall contain any general information that aids in understanding this document (e.g., background information, glossary, rationale). This section shall be divided into the following paragraphs.

4.1 Abbreviations and acronyms. This paragraph shall include an alphabetical listing of all acronyms, abbreviations, and their meanings as used in this document.

4.2 Glossary. This paragraph shall include a list of any terms and their definitions needed to understand this document. Terms often used differently between organizations (e.g., acquisition phase names, build, block, development phase names, effectivity, evolution, increment, and iteration) shall be defined to avoid confusion. If the terms used are exactly as defined in the Software Development Standard (SDSMCS), they need not be redefined here.

4.3 General information. This paragraph shall contain any other general information that aids in understanding this document (e.g., background information, rationale).

A. Appendices. Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling. Appendices shall be lettered alphabetically (Appendix A, B, etc.).

END of SMBP Template
This appendix is a MANDATORY part of the (SDSMCS). It provides the content requirements for the SMP.

1. If the section numbering used below is not used, the developer **shall** provide an appendix in the SMP with a traceability matrix mapping from the section numbers and titles below to the section numbers and titles used in the developer SMP.

2. If there is such a traceability mapping appendix, it **shall** be referenced in Section 1.3.

**Note 1:** The information shown below is consistent with the Software Development Standard for Mission Critical Systems, especially Sections 5.1 and 5.20.

**Note 2:** The definitions and terms used in this template are consistent with those in (ISO/IEC 15939).

**Note 3:** The definitions and terms used in this template are consistent with those in the Software Measurement Standard (SMS).

**Note 4:** Additional guidance is provided in the Software Measurement Standard (SMS).

**Purpose:** The Software Measurement Plan (SMP) is an integrated plan covering the software development measurement activities for all **software team members** throughout the system development. The SMP provides the planned metrics and their aggregation levels, explanations of computation, expected or projected values, thresholds, and any planned corrective actions to be taken in case thresholds are breached.

**References**


**Content Requirements**

This template contains the required content of the Software Measurement Plan (SMP). See Section 3 of the Software Development Standard for Mission Critical Systems (SDSMCS) for definitions of all italicized words or phrases.

1. **Scope.** This section **shall** be divided into the following paragraphs.

1.1 **Identification.** This paragraph **shall** contain a full identification of the system and the software to which this document applies, including, as applicable, identification number(s), title(s), abbreviation(s), version number(s), and release number(s).

1.2 **System overview.** This paragraph **shall** briefly state the purpose of the system and the software to which this document applies. It **shall:** a) describe the general nature of the system and software; b) summarize the history of system development, operation, and maintenance; c) identify the project...
sponsor, acquirer, user, developer, and support organizations; and d) identify current and planned operating and user sites.

1.3 Document overview. This paragraph shall summarize the purpose and contents of this document. This paragraph shall summarize the role of the SMP in the project’s measurement process. This paragraph shall describe any security or privacy considerations associated with its use.

1.4 Relationship to other plans and documents. This paragraph shall describe the relationship, if any, of the SMP to the SDP and other project plans and documents.

2. Referenced documents. This section shall list the number, title, revision, and date of all documents referenced in this plan. This section shall also identify the source for all documents not available through normal Government stocking activities.

3. Project measurement description. This section shall be divided into the following paragraphs.

3.1 Project measurement characteristics. This paragraph shall:
   a. Identify the project team responsible for implementing the measurement plan, including the prime contractor and other software team members;
   b. Depict and describe the organizational structure to be used for measurement, including all organizations, their relationships to one another, the authority and responsibility of each for carrying out required activities, and points of contact for each. This paragraph shall reference relevant documents with citations; and
   c. Identify current and planned operations, maintenance, and user sites.

3.2 Measurement management characteristics. This paragraph shall briefly state:
   a. The management review hierarchy for the measurement data;
   b. The management reporting systems, i.e., tools and reports, that contain measurement data; and
   c. The approval sequencing for all measurement data.

3.3 Project measurement approach. This paragraph shall be divided into paragraphs as needed to establish the context for the planning described in later sections. It shall include, as applicable, an overview of:
   a. How measurement is integrated into the technical and management processes;
   b. How system, software, and hardware measurements are related;
   c. How data will be collected and used;
   d. Measurement points of contact, i.e., prime contractor and other software team members;
   e. Measurement roles, responsibilities, and resources;
   f. Communication interfaces between project metrics personnel and organizational metrics personnel;
   g. Implementation of the measurement approach, address builds and their associated lifecycle activities, if applicable;
   h. Tools and databases to be used for the measurement process;
   i. Configuration management of the measurement process and data, including addressing how measures are added, modified, or deleted for future reports; and
   j. Evaluation criteria for the measurement process, measures, and indicators.

3.4 Software identification items. This paragraph shall provide the software identification items specified in the Software Identification Items paragraph of (SMS).
4. **Measurement description.** This section **shall** be divided into the following paragraphs. Provisions corresponding to nonrequired activities may be satisfied by the words “Not applicable.” If different builds, different software items, different part of the software items, or different types or categories of software on the project require different planning, then these differences **shall** be noted in the paragraphs. In addition to the content specified below, each paragraph **shall** identify applicable risks and uncertainties and the plans for dealing with them.

4.1 **Software measurement process.** This paragraph **shall** describe the process to be used for measurement data collection and reporting throughout the system development lifecycle. This paragraph **shall** describe the software development lifecycle model(s) to be used, including: a) planned builds, if applicable, b) their build objectives, and c) the software measurements to be collected in each build.

4.2 **General requirements.** This paragraph **shall** be divided into the following paragraphs.

4.2.1 **Measurement goals.** This paragraph **shall** be divided into the following subparagraphs.

4.2.1.1 **Organizational goals.** This paragraph **shall** describe or reference the organizational goals that impact the project’s measurement system. This paragraph **shall** discuss any organizational goals for specific process improvement initiatives either currently planned or projected.

4.2.1.2 **Project goals.** This paragraph **shall** describe or reference the project goals, quantitative or otherwise, that impact the project’s measurement system. This paragraph **shall** include any project-specific process improvement initiatives either currently planned or projected.

4.2.1.3 **Prioritized goals.** This paragraph **shall** itemize the organizational and project goals into a single priority-ordered list.

4.2.2 **Recording rationale.** This paragraph **shall** describe the approach to be followed for recording rationale for key decisions about software measurements made on the project. This paragraph **shall** interpret the term “key decisions” for the project. The rationale **shall** include tradeoffs considered, analysis methods, and criteria used to make decisions. This paragraph **shall** state where the rationale is to be recorded.

4.2.3 **Access for acquirer review.** This paragraph **shall** describe the approach to be followed for providing the acquirer and its authorized representatives access to developer and software team member quantitative data about the products and activities.

4.2.4 **Meeting contractual requirements.** This paragraph **shall** describe how the planned software measurement process covers all contractual requirements concerning software measurement collection, reporting, acquirer team access, management, and related topics. Note: The exact contractual requirements for deliverables are in the contract. Those contractual requirements state where the software measurement planning information is to be recorded. If the software measurement plan is not a deliverable, then the software measurement planning information can be included in Section 5.20 of the SDP or in a separate software measurement plan.

4.2.5 **Measurement information specification.** This paragraph **shall** contain the measurement information need description for each measure that is identified for use on the project. See
Measurement Information Specification Information Need table of the (SMS) for more information. This paragraph shall be divided into the following subparagraphs. The subsequent subparagraphs (4.2.5.x below) specify high level information about each measure. For each measure identified for use on the project, this paragraph shall provide:

4.2.5.x Measure name. This paragraph shall be divided into the following subparagraphs to describe the identified measure. Provisions corresponding to nonrequired activities may be satisfied by the words “Not applicable.”

a. Information need. What the measurement user (e.g., manager or project team member) needs to know in order to make informed decisions.
b. Information category. A logical grouping of information needs provided as structure of the measurements. The information category shall be one of the following: schedule and progress, resources and costs, product size and stability, product quality, and development performance.
c. Measurable concept. An idea for satisfying the information need by defining the data to be measured.
d. Relevant entities. The object that is to be measured. Entities include process or product elements of a project such as project tasks, plans, estimates, resources, and deliverables.
e. Base measure(s). The property or characteristic of the data that is quantified.
f. Derived measure(s). A measure that is calculated as a function of two or more base measures or other derived measures, to obtain a derived measure.
g. Prioritized goals. The list of prioritized goals, i.e., specified in Paragraph 4.2.1.3 above, to which this measure responds.

Note: Section 5 below requires detailed specifications for each indicator, base measure, and derived measure that support the measures in Paragraph 4.2.5.

5. Software measures. This section shall describe the measures to be used for software measurement throughout the system development lifecycle. This section shall also include the specific software measures to be used, i.e., collected, interpreted, analyzed, applied, reported, and used for decisionmaking, corrective actions, and reporting to the acquirer. This section shall specify which measures will be reported by lifecycle activity (e.g., requirements, design, code, integration, test). This section shall be divided into the following paragraphs. Provisions corresponding to nonrequired activities may be satisfied by the words “Not applicable.” For each measure, i.e., specified in Paragraph 4.2.5 above, selected for use on the project, this section shall provide:

5.1 Indicator specifications. This paragraph shall be divided into the following subparagraphs. For each measurement information specification identified in Paragraph 4.2.5 above, include at least one of the following:

5.1.x Indicator name. This paragraph shall be divided into the following subparagraphs to describe the identified indicator. The measurement information specification, i.e., specified in Paragraph 4.2.5 above, to which this indicator responds shall be identified.

a. Indicator description. A text discussion of how one or more measures are used to support the creation of information necessary for analysis and decision making. An indicator is often displayed as a graph or chart.
b. Example indicator diagram. A sketch of the indicator diagram incorporating sample data. This subparagraph shall provide a description of how the example indicator diagram is to be interpreted.
c. **Analysis model.** A defined process that applies decision criteria to characterize the positive or negative behavior of the indicator. If decision criteria are specified, then this field describes their use.

d. **Decision criteria.** A project performance threshold that delineates positive indicator behavior from negative indicator behavior. A defined set of actions that will be taken in response to specific values of the indicator. This paragraph shall define the responses of the measurement user to the indicator.

e. **Frequency of data analysis.** Identify how often the indicator is reported. This may be less frequently than it is collected.

f. **Responsible organization.** Identify the organization assigned to analyze the indicator and report the results.

g. **Phase of analysis.** Identify the phases or activities when the indicator is analyzed.

h. **Source of data for analysis.** Identify sources of data used in the indicator analysis.

i. **Tools used in analysis.** Identify any tools used for indicator analysis (e.g., statistical tools).

j. **User(s) of analysis.** Identify the users of the indicator results.

k. **Additional analysis guidance.** Any additional guidance on variations of this measure.

l. **Implementation considerations.** Any process or implementation requirements that are necessary for successful implementation.

5.2 **Base measure specification.** This paragraph shall be divided into the following subparagraphs. For each base measure identified in Paragraph 4.2.5 above include:

5.2.x **Base measure name.** This paragraph shall be divided into the following subparagraphs to describe the identified base measure.

a. **Measurement method.** The logical sequence of operations that define the counting rules to collect the base measure.

b. **Type of method.** The type of method used to quantify the base measure, either (1) subjective, involving human judgment, or (2) objective, using only established rules to determine numerical values.

c. **Scale.** The ordered set of values or categories used to define the base measure.

d. **Type of scale.** The type of the relationship between values on the scale, either: Nominal, Ordinal, Interval, or Range.

Note: See Measurement Information Specification Base Measure Specification table in (SMS) for more information.

e. **Unit of measure.** The standardized quantitative amount that is counted to assign value to the base measure. If tailoring is performed, the developer shall document the conversion factors between the expected standard value and the developer’s tailored value. See the Measurement Tailoring paragraph of (SMS).

Note: The developer may tailor the base measure collection and the derived measure calculations specified in the (SMS) standard, where this tailoring is limited to the use of different units of measure (UOMs) for base or derived measures.

f. **Frequency of collection.** Identify how often the data described by the base measure is collected.

g. **Responsible organization.** Identify the organization assigned to collect the base measure.

h. **Phase of collection.** Identify the phases or activities when the base measure is collected.

i. **Tools used in collection.** Identify any tools used to collect the base measure (e.g., source code analyzer).

j. **Verification and validation.** Identify any verification and validation activities (e.g., tests) that will be executed to verify that the base measure is complete and accurate.
5.3 Derived measure specification. This paragraph shall be divided into the following subparagraphs. For each derived measure identified in paragraph 4.2.5 above, include:

5.3.x Derived measure name. This paragraph shall be divided into the following subparagraphs to describe the identified derived measure.

a. Measurement function. The formula used to calculate the derived measure.

b. Scale. The ordered set of values or categories for each base measure used in the derived measure function. Valid mathematical functions are limited by base measure scale.

c. Type of scale. The type of the relationship between values on the scale for the resulting derived measure, either: Nominal, Ordinal, Interval, or Range.

d. Unit of measure. The standardized quantitative amount of the resulting derived measure.

If tailoring is performed, the developer shall document the conversion factors between the expected standard value and the developer’s tailored value. See the Measurement Tailoring paragraph of (SMS).

Note: The developer may tailor the base measure collection and the derived measure calculations specified in the (SMS) standard, where this tailoring is limited to the use of different units of measure (UOMs) for base or derived measures.

e. Frequency of calculation. Identify how often the derived measure function is calculated.

f. Responsible organization. Identify the organization assigned to perform the derived measure function calculation.

g. Phase of collection. Identify the phases or activities in which the derived measure function calculation occurs.

h. Tools used in calculation. Identify any tools used for the derived measure function calculation.

i. Verification and validation. Identify any verification and validation activities (e.g., tests) that will be executed to verify that the derived measure function calculation is complete and accurate.

6. Measurement indicator reporting and aggregation structures. This section shall be divided into the following paragraphs. Provisions corresponding to nonrequired activities may be satisfied by the words “Not applicable.”

6.1 Report aggregation. Measures reported may be aggregated to higher levels of aggregation using specific mathematical functions. Aggregation may occur up through product integration, e.g., to builds, software items, or any other applicable aggregation scheme. This paragraph shall identify the indicators, base measures, and derived measures from paragraph 5 above for which aggregation will be performed and reported. This paragraph shall specify which measures will be aggregated for each build and software item. See the Software Measurement Aggregation Considerations paragraph in (SMS) for more information on aggregation. For each indicator, base measure, and derived measure for which aggregation is to be performed, this paragraph shall provide:

6.1.1 Aggregated measures. This paragraph shall be divided into the following subparagraph to describe each identified aggregated measure.

6.1.1.x Aggregated measure name. This paragraph shall be divided into the following subparagraphs to describe the identified aggregated measure.

a. Measurement function. The formula used to calculate the aggregated measure.

b. Scale. The ordered set of values or categories for each base or derived measure used in the aggregated measure function. Valid mathematical functions are limited by base and derived measure scales.
c. **Type of scale.** The type of the relationship between values on the scale for the resulting derived measure, either: Nominal, Ordinal, Interval, or Range.

d. **Unit of measure.** The standardized quantitative amount of the resulting aggregated measure.

e. **Frequency of calculation.** Identify how often the aggregated measure function is calculated.

f. **Responsible organization.** Identify the organization assigned to perform the aggregated measure function calculation.

g. **Phase of collection.** Identify the phases or activities when the aggregated measure function calculation occurs.

h. **Tools used in calculation.** Identify any tools used in the aggregated measure function calculation.

i. **Verification and validation.** Identify any verification and validation activities (e.g., tests) that will be executed to verify that the aggregated measure function calculation is complete and accurate.

6.2 **Build aggregated measures.** This paragraph shall specify the aggregated measure names, i.e., specified in Paragraph 6.1.1 above, which will be aggregated for each build. If there are any differences in the aggregated measures between builds, then this paragraph shall specify the differences.

6.3 **Software item aggregated measures.** This paragraph shall specify the aggregated measure names, i.e., specified in Paragraph 6.1.1 above, which will be aggregated for each software item. If there are any differences in the aggregated measures between software items, then this paragraph shall specify the differences.

6.4 **Reporting.** This paragraph shall specify: a) the planned contents of measurement reports, b) frequency of measurement reports, c) planned delivery mechanism(s), i.e., electronic and human-readable), and d) planned recipients of the reports. This paragraph shall specify for each software item (SI) and build how the following four items will be reported electronically. See the Measurement Data Electronic Reporting paragraph of (SMS). This paragraph shall also specify for each software item (SI) and build how the measurement diagrams and analyses of the measurement data will be reported in human-readable form. See the Measurement Data Human-Readable Reporting paragraph of (SMS). This paragraph shall also specify the aggregated measurements to be reported: a) electronically, and b) in human-readable form. The aggregated measurements to be reported shall include:

a. **Project characteristic data.** All project characteristics, as specified in the Project Characteristics paragraph of (SMS).

b. **Base measures.** All base measures, as specified in the Base Measure Specifications appendix of (SMS), with tailored UOMs provided.

c. **Derived measures.** All derived measures, as specified in the Derived Measure Specifications appendix of (SMS), with tailored UOMs provided.

d. **Identification items.** All identification items, as specified in the Software Identification Items paragraph of (SMS).

7. **Notes.** This section shall contain any general information that aids in understanding this document (e.g., background information, glossary, rationale). This section shall be divided into the following paragraphs.

7.1 **Abbreviations and acronyms.** This paragraph shall include an alphabetical listing of all acronyms, abbreviations, and their meanings as used in this document.
7.2 Glossary. This paragraph shall include a list of any terms and their definitions needed to understand this document. Terms often used differently between organizations (e.g., acquisition phase names, build, block, development phase names, effectivity, evolution, increment, and iteration) shall be defined to avoid confusion. If the terms used are exactly as defined in the Software Development Standard (SDSMCS) and Software Measurement Standard (SMS), they need not be redefined here.

7.3 General information. This paragraph shall contain any other general information that aids in understanding this document (e.g., background information, rationale).

A. Appendices. Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling. Appendices shall be lettered alphabetically (A, B, etc.).

END of SMP Template
H.5 Software Measurement Report (SMR) Template

This appendix is a MANDATORY part of the (SDSMCS). It provides the content requirements for the SMR.

1. If the section numbering used below is not used, the developer shall provide an appendix in the SMR with a traceability matrix mapping from the section numbers and titles below to the section numbers and titles used in the developer SMR.

2. If there is such a traceability mapping appendix, it shall be referenced in Section 1.3.

Note 1: The information shown below is consistent with the Software Development Standard for Mission Critical Systems (SDSMCS), especially Sections 5.1 and 5.20.

Note 2: The software measurement report (SMR) definitions and terms used below are consistent with those in the Software Measurement Standard (SMS).

Purpose: Each Software Measurement Report (SMR) is an integrated report covering the software development activities for all significant software team members throughout the system development. The SMR provides explanations and interpretations of reported measurement data, including deviations from expected or projected values and breaches of thresholds as well as any corrective actions being undertaken. The software measurements collected and reported each month are expected to vary because the lifecycle activities vary over time.

References


Content Requirements
This template contains the required content of the Software Measurement Report (SMR). See Section 3 of the Software Development Standard for Mission Critical Systems (SDSMCS) for definitions of all italicized words or phrases.

1. Scope. This section shall be divided into the following paragraphs.

1.1 Identification. This paragraph shall contain a full identification of the system and the software to which this document applies, including, as applicable, identification number(s), title(s), abbreviation(s), version number(s), and release number(s). This paragraph shall provide the period of reporting.

1.2 System overview. This paragraph shall briefly state the purpose of the system and the software to which this document applies. It shall: a) describe the general nature of the system and software; b) summarize the history of system development, operation, and maintenance; c) identify the project

H.5-1
sponsor, acquirer, user, developer, and support organizations; and d) identify current and planned operating and user sites.

1.3 Document overview. This paragraph shall summarize the purpose and contents of this document. This paragraph shall describe any security or privacy considerations associated with its use.

1.4 Relationship to other documents and plans. This paragraph shall describe the relationship, if any, of the SMR to the Software Development Plan (SDP), software measurement plan, and other project management plans.

2. Referenced documents. This section shall list the number, title, revision, and date of all documents referenced in this plan. This section shall also identify the source for all documents not available through normal Government stocking activities.

3. Metrics analysis summary. This section shall be divided into the following paragraphs.

3.1 Contract milestones. This paragraph shall identify significant project milestones for the next six months.

3.2 Measurement performance. This paragraph shall summarize measurement performance highlights that are detailed in the subsequent report. When a threshold identified in the Software Measurement Plan (SMP) is breached, then mitigation planning shall be itemized in this section.

4. General requirements. This section shall be divided into the following paragraphs. Provisions corresponding to nonrequired activities may be satisfied by the words “Not applicable.” If different builds or different software on the project require different planning, these differences shall be noted in the paragraphs.

4.1 Project build and software item characteristics. For each build and software item (SI), this paragraph shall provide the context information specified in the Project Characteristics paragraph of the Software Measurement Standard (SMS):
   a. Computer resource characterization;
      (1) Computer hardware identification;
      (2) Computer communication identification;
      (3) Computer storage hardware identification;
   b. Authorizing agreement (e.g., Memorandum of Understanding (MOU), contract, amendment);
   c. Development organization(s);
   d. Capability Maturity Model Integration for Development (CMMI®-DEV) maturity level;
   e. Application type;
   f. Development process;
   g. Software origin;
   h. Computer language;
   i. Reusable software applications, including COTS and acquirer-furnished software.

4.1.x Build x <Insert Name> characteristics. For each build “x,” this paragraph shall provide the context information specified in paragraph 4.1 above. If any of these characteristics changes, then this paragraph shall highlight the change(s).
4.1.x.y  **Software item x.y <Insert Name> characteristics.** For each *software item (SI)* “y” in the *build*, this paragraph **shall** provide the context information specified in paragraph 4.1 above. If any of these characteristics changes, then this paragraph **shall** highlight the change(s).

4.2  **Identification items.** For each *build* and *SI*, this paragraph **shall** provide the identification information specified in the Software Identification Items paragraph of (SMS):

a. WBS element identifier;
b. Control account;
c. Integrated master schedule (IMS);
d. Specification tree identification number;
e. Component identification;
f. Acquisition phase identification;
g. Development phase identification;
h. System identification (version and release);
i. Subsystem identification (version and release);
j. Element identification (version and release);
k. Software item identification (version and release);
l. Build identification (Release);
m. Computer resource utilization (CPU):
   (1) CPU identification(s) (ID(s));
   (2) Computer resource utilization input/output ID;
   (3) Computer resource utilization memory storage device;

n. Computer resource utilization response time.

4.2.x  **Build x <Insert Name> identification items.** For each *build*, this paragraph **shall** provide the context information specified in paragraph 4.2 above. If any of these characteristics changes, then this paragraph **shall** highlight the change(s).

4.2.x.y  **Software item x.y <Insert Name> identification items.** For each *SI* in the *build*, this paragraph **shall** provide the context information specified in paragraph 4.2 above. If any of these characteristics changes, then this paragraph **shall** highlight the change(s).

5.  **Measurement data human-readable reporting.** This section **shall** report the data by the particular component, e.g., *build*, increment, or evolution, to which they apply. In addition, this section **shall** report the data by the *SI* to which they apply. This section **shall** be divided into the following paragraphs. Provisions corresponding to nonrequired activities may be satisfied by the words “Not applicable.” For each measurement diagram whenever a tailored Unit of Measure (UOM) is used, a footnote or other appropriate notation **shall** document that fact. For each measurement diagram, the subsections **shall** provide the following labeling information specified in the Measurement Diagram Content and Labeling paragraph of (SMS):

a. Scope of the data;
b. *SI* or *build* name;
c. Product integration level;
d. Reporting period;
e. Reported by.

5.x  **Build identifier.** Identify the specific component, e.g., *build*, increment, or evolution, to which each of the subsequent measurements apply. This paragraph, including each of its sub-items and its aggregated data (5.x.y through 5.x.y+1), **shall** be repeated for each *build* “x.”
5.x.y  **Software item identifier.** Identify the specific software item (SI) to which each of the subsequent measurements apply. In addition to the content specified below, each paragraph shall identify applicable risks and uncertainties and the plans for dealing with them. This paragraph, including each of its sub-items and its aggregated data (5.x.y through 5.x.y.18, where “.x” is the build and “.y” is the software item) shall be repeated for each software item “y.”

5.x.y.1  **Requirement progress management indicator.** For the current reporting period, this paragraph shall provide in tabular and graphic form the following base and derived measures of requirement progress. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Requirements progress management paragraph:
   a. Requirements defined
   b. Requirements TBX closure
   c. Requirements verified
   d. Qualification methods

5.x.y.2  **Development progress management indicator.** For the current reporting period, this paragraph shall provide in tabular and graphic form the following base and derived measures of development progress. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Development progress management paragraph:
   a. Components defined
   b. Units defined
   c. Units coded and unit tested
   d. Units integrated and tested

5.x.y.3  **Test progress management indicator.** For the current reporting period, this paragraph shall provide in tabular and graphic form the following base and derived measures of test progress. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Test progress management paragraph:
   a. Test cases developed
   b. Test cases dry run
   c. Test cases performed
   d. Test cases passed

5.x.y.4  **Schedule adherence management indicator.** For the current reporting period, this paragraph shall provide in tabular and graphic form the following base and derived measures of schedule adherence. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Schedule adherence management paragraph:
   a. Project milestones
   b. Scheduled activities

5.x.y.5  **Effort profile management indicator.** For the current reporting period, this paragraph shall provide in tabular and graphic form the following base and derived measures of effort profile management. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Effort profile management paragraph:
   a. Labor hours
   b. Rework labor hours
5.x.y.6  **Staff profile management indicator.** For the current reporting period, this paragraph **shall** provide in tabular and graphic form the following base and derived measures of staff profile management. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Staff profile management paragraph:
   a. Staffing level
   b. Staff by experience
   c. Staff turnover

5.x.y.7  **Computer resource management indicator.** For the current reporting period, this paragraph **shall** provide in tabular and graphic form the following base and derived measures of computer resource management. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Computer resource management paragraph:
   a. CPU utilization
   b. Memory utilization
   c. Input/output utilization
   d. Response time

5.x.y.8  **Cost profile management indicator.** For the current reporting period, this paragraph **shall** provide in tabular and graphic form the following base and derived measures of cost profile management. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Cost profile management paragraph:
   a. Earned value performance
   b. Schedule and cost performance index
   c. Schedule and cost variance

5.x.y.9  **Size management indicator.** For the current reporting period, this paragraph **shall** provide in tabular and graphic form the following base and derived measures of size management. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Size management paragraph:
   a. Requirements size
   b. Requirements by type
   c. Line of code size
   d. Line of code by origin
   e. Line of code by type

5.x.y.10  **Volatility management indicator.** For the current reporting period, this paragraph **shall** provide in tabular and graphic form the following base and derived measures of volatility management. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Volatility management paragraph:
   a. Requirement volatility
   b. Line of code volatility

5.x.y.11  **Build content management indicator.** For the current reporting period, this paragraph **shall** provide in tabular and graphic form the following base measures of build content management. For the current reporting period, plot and report base measures in accordance with the instructions provided in the (SMS) Build content management paragraph:
   a. Requirements per build

5.x.y.12  **Defect resolution management indicator.** For the current reporting period, this paragraph **shall** provide in tabular and graphic form the following base and derived measures of defect
resolution management. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Defect resolution management paragraph:
   a. Discrepancy report status
   b. Discrepancy report aging
   c. Discrepancy report by type
   d. Discrepancy report by source

5.x.y.13  Complexity management indicator. For the current reporting period, this paragraph shall provide in tabular and graphic form the following base measures of complexity management. For the current reporting period, plot and report the base measure in accordance with the instructions provided in the (SMS) Complexity management paragraph:
   a. Cyclomatic complexity

5.x.y.14  Coverage management indicator. For the current reporting period, this paragraph shall provide in tabular and graphic form the following base and derived measures of coverage management. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Coverage management paragraph:
   a. Requirements to design traceability
   b. Requirements to test traceability

5.x.y.15  Productivity management indicator. For the current reporting period, this paragraph shall provide in tabular and graphic form the following base and derived measures of productivity management. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Productivity management paragraph:
   a. Development productivity

5.x.y.16  Maturity management indicator. For the current reporting period, this paragraph shall provide in tabular and graphic form the following base and derived measures of maturity management. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Maturity management paragraph:
   a. Development defect density

5.x.y.17  Management status management indicator. For the current reporting period, this paragraph shall provide in tabular and graphic form the following base and derived measures of management status. For the current reporting period, plot and report base and derived measures in accordance with the instructions provided in the (SMS) Management status paragraph:
   a. Action item closure
   b. Risk mitigation task Completion Status
   c. Schedule compression

5.x.y.18  Aggregated measurement report (SI level). For the current reporting period, this paragraph shall report any aggregated measurements identified for the project as specified in the project software measurement plan (SMP) for the software item (SI). This information shall be reported for each paragraph 5.x.y SI after all of the other measurements for that SI.

5.x.y+1  Aggregated measurement report (build level). For the current reporting period, this paragraph shall report any aggregated measurements identified for the project as specified in the project software measurement plan (SMP) for the build. This information shall be reported for each paragraph 5.x build after all of the other measurements of the software items for that build.
6. **Measurement data electronic reporting.** This section **shall** specify the electronic reporting of measurement data. This section **shall** be divided into the following paragraphs.

6.1 **Base measures.** In accordance with the data definitions provided in (SMS) Appendix A, Base Measure Specifications, this paragraph **shall** provide the planned base measure data. For the current reporting period, this paragraph **shall** provide actual base measure counts in accordance with the data definition provided in (SMS) the Base Measure Specifications appendix.

6.2 **Derived measures.** For the current reporting period, this paragraph **shall** report the calculated derived values in accordance with the data definition provided in (SMS) the Derived Measure Specifications appendix.

6.3 **Aggregated measurement report.** Report any aggregated measurements identified for the project as specified in the project software measurement plan (SMP).

7. **Notes.** This section **shall** contain any general information that aids in understanding this document (e.g., background information, glossary, rationale). This section **shall** be divided into the following paragraphs.

7.1 **Abbreviations and acronyms.** This paragraph **shall** include an alphabetical listing of all acronyms, abbreviations, and their meanings as used in this document.

7.2 **Glossary.** This paragraph **shall** include a list of any terms and their definitions needed to understand this document. Terms often used differently between organizations (e.g., acquisition phase names, build, block, development phase names, effectivity, evolution, increment, and iteration) **shall** be defined to avoid confusion. If the terms used are exactly as defined in the Software Development Standard (SDSMCS) and Software Measurement Standard (SMS), they need not be redefined here.

7.3 **General information.** This paragraph **shall** contain any other general information that aids in understanding this document (e.g., background information, rationale).

A. **Appendices.** Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each appendix **shall** be referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling. Appendices **shall** be lettered alphabetically (A, B, etc.).

END of SMR Template
H.6 Process Improvement Plan (PIP) Template

This appendix is a MANDATORY part of the (SDSMCS). It provides the content requirements for the Process Improvement Plan (PIP).

1. If the section numbering used below is not used, the developer shall provide an appendix in the PIP with a traceability matrix mapping from the section numbers and titles in the PIP template below to the section numbers and titles used in the developer PIP.

2. If there is such a traceability mapping appendix, it shall be referenced in Section 1.3.

Note 1: The information below is consistent with the Software Development Standard for Mission Critical Systems (SDSMCS), especially Section 5.25.

Purpose: The PIP is an integrated plan covering the software development process improvement activities for all software team members throughout the system development life cycle. The PIP provides the plans and activities to improve the processes on the project based on process assessments conducted by the acquirer team or the developer team or both.

References


Content Requirements

This template contains the required content of the PIP. See Section 3 of the Software Development Standard for Mission Critical Systems (SDSMCS) for definitions of all italicized words or phrases.

1. Scope. This section shall be divided into the following paragraphs.

1.1 Identification. This paragraph shall contain a full identification of the system and the software to which this document applies, including, as applicable, identification number(s), title(s), abbreviation(s), version number(s), and release number(s).

1.2 System overview. This paragraph shall briefly state the purpose of the system and the software to which this document applies. This paragraph shall: a) describe the general nature of the system and software; b) summarize the history of system development, operation, and maintenance; c) identify the project sponsor, acquirer, user, developer, and support organizations; and d) identify current and planned operating and user sites.

1.3 Document overview. This paragraph shall summarize the purpose and contents of this document. This paragraph shall describe any security or privacy considerations associated with its use.
1.4 **Relationship to other documents and plans.** This paragraph shall describe the relationship, if any, of the PIP to the software development plan, software measurement plan, and other project management plans.

2. **Referenced documents.** This section shall list the number, title, revision, and date of all documents referenced in this plan. This section shall also identify the source for all documents not available through normal Government stocking activities.

3. **Process improvement background.** This section shall provide background on the project’s processes and process improvement efforts.

3.1 **Process baselines.** This paragraph shall include the process title, process identifier, organization site, and date of each process in each identified baseline below:

- Identified process baseline(s) from which process improvements are to be made; and
- Identified process baseline(s) of each software team member, if any, that use different processes from those defined for the project or levied by the prime contractor on the other software team members.

3.2 **Process appraisal results.** For each process appraisal or process audit performed on the project on any member of the developer team, including the prime contractor and other software team members, this paragraph shall include:

- Results including dates, detailed final findings, and whether from project self-appraisal(s), acquirer appraisal(s), or combined acquirer and developer appraisal team(s);
- The appraisal disclosure statements (ADSs) and any additional clarifying materials from all SCAMPIs, if any, performed; and
- The current resolution status of the findings.

3.3 **Previous process improvements.** This paragraph shall include:

- Descriptions of previous process improvement efforts that have been performed on the project;
- Other process improvement efforts, if any, that apply to the project;
- Any additional clarifying information related to process improvement;
- The project’s process improvements implemented so far; and
- List of process improvement artifacts.

4. **Process improvement goals and success criteria.** This paragraph shall:

- Identify the goals and objectives for process improvement;
- Describe the project’s planned process improvements;
- Define the process improvement task success criteria;
- Define how these criteria are measured;
- Identify process improvement measurements; and
- For the project software team members, if any, that are part of CMMI high maturity organizations, identify the quantitative success criteria for evaluating the results.10

---

8Standard CMMI Appraisal Method for Process Improvement (SCAMPI®)
9High-maturity organizations are appraised by an Appraisal Requirements for CMMI (ARC) Class A SCAMPI appraisal as practicing and successfully rated by an independent CMMI Institute-certified High Maturity Lead Appraiser, as Maturity Level 4 or 5, including all 20 or 22 CMMI-DEV V1.3 process areas, respectively. The Level 4 CMMI high-maturity process areas are: Organizational Process Performance (OPP) and Quantitative Project Management (QPM); the Level 5 process areas are: Causal Analysis and Resolution (CAR) and Organizational Performance Management (OPM).
10CMMI® and SCAMPI® are a trademark and service mark of Carnegie Mellon University.
5. **Project process improvement organization.** This section **shall** depict and describe the organizational structure to be used for process improvement on the project. Note: Reference Section 7.1 of the SDP for the project organization.

5.1 **Process improvement groups.** This paragraph **shall:**
   a. Describe the project’s process improvement groups, i.e., the enduring groups;
   b. Describe the process action teams, i.e., short-duration teams for developing or improving specific *processes*;
   c. Provide charters that identify the focus of the activities and the responsibilities for each process improvement group and process action team.

5.2 **Process improvement processes and procedures.** This paragraph **shall:**
   a. Include the *processes* and procedures to be used to manage the project’s process improvement activities;
   b. Identify the checklists, templates, and work instructions for performing and managing the improvement tasks and generating the resulting *work products*;
   c. Identify the necessary management, development, and support activities for process improvement; and
   d. Identify the planned process improvement *work products*.

5.3 **Project process architecture.** This paragraph **shall** provide the process architecture across the *prime contractor* and all *software team members*. It shall include inputs, outputs, sequencing, interfaces, interdependencies, and other relationships between the *processes* and procedures in the project’s defined *processes* and any other relevant processes (e.g., corporate *processes*).

6. **Process improvement planning.** This section **shall** describe the planning of the process improvement activities. The process improvement planning **shall:**
   a. Include any other inputs, besides the findings in Paragraph 3.2, to the process improvement plan (e.g., lessons learned, process effectiveness measures);
   b. Describe the process improvement activities that address the findings;
   c. Identify any 1) barriers and *risks* to implementing this plan, and 2) the risk management strategy appropriate for each of them;
   d. Describe how this process improvement plan will be applied to the *software team members* performing *software development* on the project; and
   e. Specify the effort, budget, schedule, and other resources for the process improvement activities, including the basis of estimates and assumptions made.

7. **Process improvement implementation and tracking.** This section **shall** describe the steps to be followed in:
   a. Implementing this plan;
   b. Tracking progress against this plan; and
   c. Measuring the effectiveness of the process improvement activities, including measurements and quality assurance activities.

8. **Notes.** This section **shall** contain any general information that aids in understanding this *document* (e.g., background information, glossary, rationale). This section **shall** be divided into the following paragraphs.
8.1 Abbreviations and acronyms. This paragraph shall include an alphabetical listing of all acronyms, abbreviations, and their meanings as used in this document.

8.2 Glossary. This paragraph shall include a list of any terms and their definitions needed to understand this document. Terms often used differently between organizations (e.g., acquisition phase names, build, block, development phase names, effectivity, evolution, increment, and iteration) shall be defined to avoid confusion. If the terms used are exactly as defined in the Software Development Standard (SDSMCS), they need not be redefined here.

8.3 General information. This paragraph shall contain any other general information that aids in understanding this document (e.g., background information, rationale).

A. Appendices. Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling. Appendices shall be lettered alphabetically (A, B, etc.).
Appendix I. Document Improvement Proposal

I.1 Recommended Changes

Suggestions for changes, i.e., changes, additions, deletions, to this standard should be in the form of a proposed change of text, together with appropriate supporting rationale and any comments or data which can be of use in improving this document. These change suggestions can be submitted using the Recommendation for Change form appearing at the end of this document. Send the Recommendation for Change form to:

Director, Software Acquisition and Process Department (SAPD)
The Aerospace Corporation
P. O. Box 2957 – M1/112
Los Angeles, CA 90009-2957

A comment form suitable for a series of comments is available from SAPD.
**Document Improvement Proposal**

**INSTRUCTIONS**

1. Complete blocks 1 through 7. All blocks must be completed.
2. Send to the Preparing Organization specified in block 8.

NOTE: Do not use this form to request copies of documents, or to request waivers, or clarification of requirements on current contracts. Comments submitted on this form do not constitute or imply authorization to waive any portion of the referenced document(s) or to amend contractual requirements. Comments submitted on this form do not constitute a commitment by the Preparing Activity to implement the suggestion; the Preparing Authority will coordinate a review of the comment and provide disposition to the comment submitter specified in Block 6.

<table>
<thead>
<tr>
<th>DOCUMENT CHANGE RECOMMENDATION:</th>
<th>1. Document Number:</th>
<th>2. Document Date:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TR-RS-2015-00012</td>
<td></td>
</tr>
</tbody>
</table>

|--------------------|-----------------------------------------------------------|

<table>
<thead>
<tr>
<th>4. Nature of Change:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify paragraph number, and requirement identifier, if applicable; include proposed revision language and supporting data. Attach extra sheets as needed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5. Reason for Recommendation:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6. Submitter Information:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Name:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>c. Address:</td>
</tr>
<tr>
<td>e. E-mail address:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8. Preparing Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Aerospace Corporation</td>
</tr>
<tr>
<td>P.O. Box 92957 – M1/112</td>
</tr>
<tr>
<td>El Segundo, CA  90009</td>
</tr>
<tr>
<td>Attention: Director, Software Acquisition and Process Department</td>
</tr>
</tbody>
</table>

2014
Software Development Standard for Mission Critical Systems

Approved Electronically by:

David J. Gorney, EXECUTIVE VP
OFFICE OF EVP/SSG

© The Aerospace Corporation, 2014.
All trademarks, service marks, and trade names are the property of their respective owners.
SL0113
Software Development Standard for Mission Critical Systems

Approved by:

Andrew B. Dawdy, Principal Director
Engineering Directorate
Engineering and Integration Division
Space Systems Group

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

Frank L. Knight, Systems Director
Engineering
Engineering and Integration Division
Space Systems Group

Leslie J. Holloway, Director
Software Acquisition and Process Department
Computers and Software Division
Engineering and Technology Group

© The Aerospace Corporation, 2014.

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

**REPORT TITLE**

Software Development Standard for Mission Critical Systems

<table>
<thead>
<tr>
<th>REPORT NO.</th>
<th>PUBLICATION DATE</th>
<th>SECURITY CLASSIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR-RS-2015-00012</td>
<td>March 17, 2014</td>
<td>UNCLASSIFIED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Department/Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thomas Fitzgerald</td>
<td><a href="mailto:Thomas.fitzgerald.5@us.af.mil">Thomas.fitzgerald.5@us.af.mil</a></td>
<td>Space and Missile Systems Center</td>
</tr>
<tr>
<td>Michael Hinshaw</td>
<td><a href="mailto:Michael.hinshaw@us.af.mil">Michael.hinshaw@us.af.mil</a></td>
<td>Space and Missile Systems Center</td>
</tr>
<tr>
<td>Maj. Mark Silvanius</td>
<td><a href="mailto:silvaniusm@nro.mil">silvaniusm@nro.mil</a></td>
<td>National Reconnaissance Office</td>
</tr>
<tr>
<td>David E. Davis</td>
<td><a href="mailto:David.davis.3@us.af.mil">David.davis.3@us.af.mil</a></td>
<td>Space and Missile Systems Center</td>
</tr>
<tr>
<td>Thomas Meyers</td>
<td><a href="mailto:Thomas.meyers@us.af.mil">Thomas.meyers@us.af.mil</a></td>
<td>Space and Missile Systems Center</td>
</tr>
<tr>
<td>Jeff Mayer</td>
<td><a href="mailto:Jeffrey.mayer@us.af.mil">Jeffrey.mayer@us.af.mil</a></td>
<td>USAF Materiel Command</td>
</tr>
<tr>
<td>Naim (Nick) Awwad</td>
<td><a href="mailto:Naim.awwad@us.af.mil">Naim.awwad@us.af.mil</a></td>
<td>Space and Missile Systems Center</td>
</tr>
<tr>
<td>Mark Oja</td>
<td><a href="mailto:Mark.oja@atk.com">Mark.oja@atk.com</a></td>
<td>ATK</td>
</tr>
<tr>
<td>David Eller</td>
<td><a href="mailto:David.eller@us.af.mil">David.eller@us.af.mil</a></td>
<td>USAF ICBM Office</td>
</tr>
<tr>
<td>James A. Haag</td>
<td><a href="mailto:James.haag@us.af.mil">James.haag@us.af.mil</a></td>
<td>Space and Missile Systems Center</td>
</tr>
<tr>
<td>John Lindberg</td>
<td><a href="mailto:jon.lindberg@jhuapl.edu">jon.lindberg@jhuapl.edu</a></td>
<td>The Johns Hopkins University</td>
</tr>
<tr>
<td>Patrick Lehnerd</td>
<td><a href="mailto:Pat.lehnerd@atk.com">Pat.lehnerd@atk.com</a></td>
<td>ATK Aerospace Systems Group</td>
</tr>
<tr>
<td>Nancy Droz</td>
<td><a href="mailto:Nancy.droz@us.af.mil">Nancy.droz@us.af.mil</a></td>
<td>Space and Missile Systems Center</td>
</tr>
<tr>
<td>Amit Patel</td>
<td><a href="mailto:amit.m.patel@lmco.com">amit.m.patel@lmco.com</a></td>
<td>SBIRS Chief Software Engineer</td>
</tr>
<tr>
<td>Sherri Fike</td>
<td><a href="mailto:sfike@ball.com">sfike@ball.com</a></td>
<td>Ball Aerospace &amp; Technologies Corp.</td>
</tr>
<tr>
<td>Name</td>
<td>Organization</td>
<td>Email</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>Wanda Sigur</td>
<td>Lockheed Martin Space Systems Co.</td>
<td><a href="mailto:Wandaanne.a.sigur@lmco.com">Wandaanne.a.sigur@lmco.com</a></td>
</tr>
<tr>
<td>Beth Emery</td>
<td>Northrop Grumman Corporation</td>
<td><a href="mailto:Beth.emery@ngc.com">Beth.emery@ngc.com</a></td>
</tr>
<tr>
<td>Dave Swanson</td>
<td>Orbital Sciences Corporation</td>
<td><a href="mailto:Swanson.david@orbital.com">Swanson.david@orbital.com</a></td>
</tr>
<tr>
<td>John Nelson</td>
<td>Lockheed Martin Space Systems Co.</td>
<td><a href="mailto:John.d.nelson@lmco.com">John.d.nelson@lmco.com</a></td>
</tr>
<tr>
<td>Trisha Hinters</td>
<td>Northrop Grumman Corporation</td>
<td><a href="mailto:Trisha.hinners@ngc.com">Trisha.hinners@ngc.com</a></td>
</tr>
<tr>
<td>Ben Hoang</td>
<td>Orbital Sciences Corporation</td>
<td><a href="mailto:Hoang.ben@orbital.com">Hoang.ben@orbital.com</a></td>
</tr>
<tr>
<td>Larry Patzman</td>
<td>Lockheed Martin Space Systems Co.</td>
<td><a href="mailto:Laurence.s.patzman@lmco.com">Laurence.s.patzman@lmco.com</a></td>
</tr>
<tr>
<td>Michael Mantz</td>
<td>Northrop Grumman Corporation</td>
<td><a href="mailto:Michael.mantz@ngc.com">Michael.mantz@ngc.com</a></td>
</tr>
<tr>
<td>Barry Johnson</td>
<td>Orbital Sciences Corporation</td>
<td><a href="mailto:Johnson.barry@orbital.com">Johnson.barry@orbital.com</a></td>
</tr>
<tr>
<td>Harry Lockwood</td>
<td>Lockheed Martin Space Systems</td>
<td><a href="mailto:harry.lockwood@lmco.com">harry.lockwood@lmco.com</a></td>
</tr>
<tr>
<td>Ruth Bishop</td>
<td>Northrop Grumman Corporation</td>
<td><a href="mailto:Ruth.bishop@ngc.com">Ruth.bishop@ngc.com</a></td>
</tr>
<tr>
<td>Rich Patrican</td>
<td>Raytheon Company, Space and Airborne Systems</td>
<td><a href="mailto:Richard.A.Patrican@raytheon.com">Richard.A.Patrican@raytheon.com</a></td>
</tr>
<tr>
<td>Steve Trinh</td>
<td>DCMA Space &amp; Missile Systems Division</td>
<td><a href="mailto:Steven.trinh@losangeles.af.mil">Steven.trinh@losangeles.af.mil</a></td>
</tr>
<tr>
<td>James Wade</td>
<td>MIT Lincoln Laboratory</td>
<td><a href="mailto:jwade@ll.mit.edu">jwade@ll.mit.edu</a></td>
</tr>
<tr>
<td>Mark Baldwin</td>
<td>Raytheon Company, Space and Airborne Systems</td>
<td><a href="mailto:Mark_L_baldwin@raytheon.com">Mark_L_baldwin@raytheon.com</a></td>
</tr>
<tr>
<td>Paul Hopkins</td>
<td>Lockheed Martin Space Systems Co.</td>
<td><a href="mailto:Paul.e.hopkins@lmco.com">Paul.e.hopkins@lmco.com</a></td>
</tr>
<tr>
<td>Kristen Baldwin</td>
<td>Office of the Deputy Assistant Secretary of Defense for Systems Engineering (ODASD(SE))</td>
<td><a href="mailto:Kristen.baldwin@osd.mil">Kristen.baldwin@osd.mil</a></td>
</tr>
<tr>
<td>Steve Lowell</td>
<td>Defense Standardization Program Office</td>
<td><a href="mailto:Stephen.Lowell@dla.mil">Stephen.Lowell@dla.mil</a></td>
</tr>
<tr>
<td>Craig Wesser</td>
<td>Northrop Grumman Corporation</td>
<td><a href="mailto:Craig.wesser@ngc.com">Craig.wesser@ngc.com</a></td>
</tr>
<tr>
<td>John McBride</td>
<td>Orbital Sciences Corporation</td>
<td><a href="mailto:McBride.john@orbital.com">McBride.john@orbital.com</a></td>
</tr>
<tr>
<td>Darlene Mosser-Kerner</td>
<td>Office of the Deputy Assistant Secretary of Defense for Developmental Test and Evaluation</td>
<td><a href="mailto:Darlene.s.mosserkerner.ctr@mail.mil">Darlene.s.mosserkerner.ctr@mail.mil</a></td>
</tr>
</tbody>
</table>