AEROSPACE REPORT NO.
TOR-2011(8591)-20

Space Segment Software Readiness Assessment

June 3, 2011

Suellen Eslinger[1], Leslie J. Holloway[2], and Robyn Wilkes[2]
[1]Software Engineering Subdivision, Computers and Software Division
[2]Software Acquisition and Process Department, Software Engineering Subdivision

Prepared for:

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

Contract No. FA8802-09-C-0001

Authorized by: Engineering and Technology Group

Developed in conjunction with Government and Industry contributions as part of the U.S. Space Programs Mission Assurance Improvement workshop.

AEROSPACE
Assuring Space Mission Success

AEROSPACE REPORT NO.
TOR-2011(8591)-20

Space Segment Software Readiness Assessment

June 3, 2011

Suellen Eslinger[1], Leslie J. Holloway[2], and Robyn Wilkes[2]
[1]Software Engineering Subdivision, Computers and Software Division
[2]Software Acquisition and Process Department, Software Engineering Subdivision

Prepared for:

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

Contract No. FA8802-09-C-0001

Authorized by: Engineering and Technology Group

Developed in conjunction with Government and Industry contributions as part of the U.S. Space Programs Mission Assurance Improvement workshop.

AEROSPACE REPORT NO.
TOR-2011(8591)-20

Space Segment Software Readiness Assessment

June 3, 2011

Suellen Eslinger[1], Leslie J. Holloway[2], and Robyn Wilkes[2]
[1]Software Engineering Subdivision, Computers and Software Division
[2]Software Acquisition and Process Department, Software Engineering Subdivision

Prepared for:

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

Contract No. FA8802-09-C-0001

Authorized by: Engineering and Technology Group

Developed in conjunction with Government and Industry contributions as part of the U.S. Space Programs Mission Assurance Improvement workshop.

AEROSPACE
Assuring Space Mission Success

AEROSPACE REPORT NO.
TOR-2011(8591)-20

Space Segment Software Readiness Assessment

Approved by:

Malina M Hills

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

Malina Hills, General Manager
MILSATCOM Division
Space Programs Operations
Space Systems Group

Acknowledgments

This document has been produced as a collaborative effort of the Mission Assurance Improvement Workshop. The forum was organized to enhance mission assurance processes and supporting disciplines through collaboration between industry and government across the U.S. Space Program community utilizing an issues-based approach. The approach is to engage the appropriate subject matter experts to share best practices across the community in order to produce valuable mission assurance guidance documentation.

The document was created by multiple authors throughout the government and the aerospace industry. We thank the following contributing authors for making this collaborative effort possible:

Claudia Baker (Boeing)
Bonnie Triezenberg (Boeing)
Robin Fulford (General Dynamics)
Steven Pereira (Johns Hopkins University Applied Physics Laboratory)
Dwight Wrench (Lockheed Martin Corporation)
Greg Whittaker (The Mitre Corporation)
Cassie Weissert (Orbital Sciences Corporation)
Michael Ostroff (Orbital Sciences Corporation)
Kilian Chan (Northrop Grumman Aerospace Systems)
Cathy Karnes (Raytheon Space and Airborne Systems)

A special thank-you for co-leading this team and efforts to ensure completeness and quality of this document goes to:

Michael Ostroff and Cassie Weissert (Orbital Sciences Corporation)
Suellen Eslinger (The Aerospace Corporation)

The Software Readiness Assessment Topic Team would like to acknowledge the support, contributions, and feedback from the following organizations:

The Aerospace Corporation
Ball Aerospace & Technologies Corp
The Boeing Company
General Dynamics
Johns Hopkins University Applied Physics Laboratory
Lockheed Martin Corporation
Northrop Grumman Aerospace Systems
Orbital Sciences Corporation
Raytheon Space and Airborne Systems
The Mitre Corporation
Space and Missile Systems Center (SMC)

The authors deeply appreciate the contributions of the software subject matter experts who reviewed the document:

Melanie Christensen (Ball Aerospace and Technologies Corporation)
Dorothy McKinney (Lockheed Martin Corporation)
Tony D’Agostino and Ed Wild (Northrop Grumman Aerospace Systems)
Christina Tudor (Orbital Sciences Corporation)
Raul Ponce (Pratt and Whitney Rocketdyne [PWR])
Bruce Wetts (Raytheon Space and Airborne Systems)
Jay Fisher (United Launch Alliance, LLC [ULA])

Executive Overview

The Mission Assurance Improvement Workshop steering committee selected space segment software readiness assessment as a topic for 2011 because of the increasing importance of software in space systems. The acquisition of space systems has recently been fraught with problems due to software, up to and including mission failure. The steering committee decided that more emphasis on software mission assurance is needed in the development of space system mission-critical software, especially in the early life cycle phases.

The mechanism selected for increasing space segment software mission assurance for this year's study is the software readiness assessment. A software readiness assessment is an evaluation of software technical and management maturity at predefined points in the software product life cycle. The purpose of a software readiness assessment is to determine the readiness of the software for the next phase of development or for an upcoming designated event and the associated risk of proceeding. When software products move to the next phase of the development life cycle without achieving sufficient maturity, the result is that defects are passed into downstream activities where they are less likely to be found, cost more, and take more time to fix. The goal of a software readiness assessment is to provide objective, independent feedback to the program's software stakeholders who can determine if the mission assurance requirements of the system have been met.

The space segment software readiness assessment team was made up of representatives from The Aerospace Corporation, The Boeing Company, General Dynamics Corporation, Johns Hopkins University Applied Physics Laboratory, Lockheed Martin Company, The MITRE Corporation, Northrop Grumman Aerospace Systems, Orbital Sciences Corporation, and Raytheon Space and Airborne Systems. The team met early in the fiscal year and defined a task plan for the development of a document. The team met weekly by telecom and monthly face-to-face. The team unanimously determined the scope and content of the document and all team members contributed to the document development.

This document is the output of the team's effort. It defines a process for conducting software readiness assessments and a set of criteria for evaluating software readiness. The use of this document is intended to ensure the completeness and uniformity of assessments across different assessment teams, on different programs, and at different points in the life cycle. The team members intend for this document to be a useful tool to organizations developing space segment software.

Contents

Figures

Tables

1. Introduction

1.1 Objective

The acquisition of space systems is often fraught with problems due to software-related issues, including performance deficiencies, software defects, on-orbit anomalies, cost or schedule overruns, and launch delays. When software products move to the next phase of the development life cycle without achieving sufficient maturity, the result is that defects are passed into downstream activities where they cost more and take more time to fix. Clearly, more emphasis on software mission assurance is needed in the development of space system mission-critical software, especially in the early life cycle phases.

A specific software mission assurance activity that has been found to be very effective on space programs is a software readiness assessment (SRA). An SRA is an evaluation of software technical and management maturity at predefined points in the software product life cycle. The purpose of an SRA is to determine the readiness of the software for the next phase of development or for an upcoming designated event and the associated risk of proceeding. The ultimate goal is to provide objective, independent feedback to the program's software stakeholders.

This document defines a process for conducting SRAs and a set of assessment criteria for evaluating software readiness. The use of this document is intended to ensure the completeness and uniformity of assessments across different assessment teams, on different programs, and at different points in the life cycle.

In performing an SRA, the SRA team evaluates risk by assessing software planning, technical performance, execution progress, and software quality, and their effect on software cost and schedule. To accomplish this, the SRA team applies the assessment criteria provided in this document to three perspectives of the software development program as it evolves through the life cycle: products, processes, and resources.

1.2 Scope

The scope of the SRAs defined in this document is limited to the space segment software. A generic structure of a space system program is shown in Figure 1. This figure uses dark blue to show the space segment software areas that are in the scope of the SRAs described in this document. This includes all onboard software, including spacecraft bus software and payload software. In addition, the space segment software includes the software in a satellite operations center that is used for satellite command and control (e.g., telemetry processing, tracking, commanding, satellite health and status, and satellite operations planning and execution). The space segment software does not include other ground operations and ground support software (e.g., collection management, asset allocation, mission data processing, data dissemination, ground control, and network control) or user equipment software.

Although this document is limited to addressing these areas of space system software, most of the assessment criteria apply to any mission-critical software with high assurance requirements.

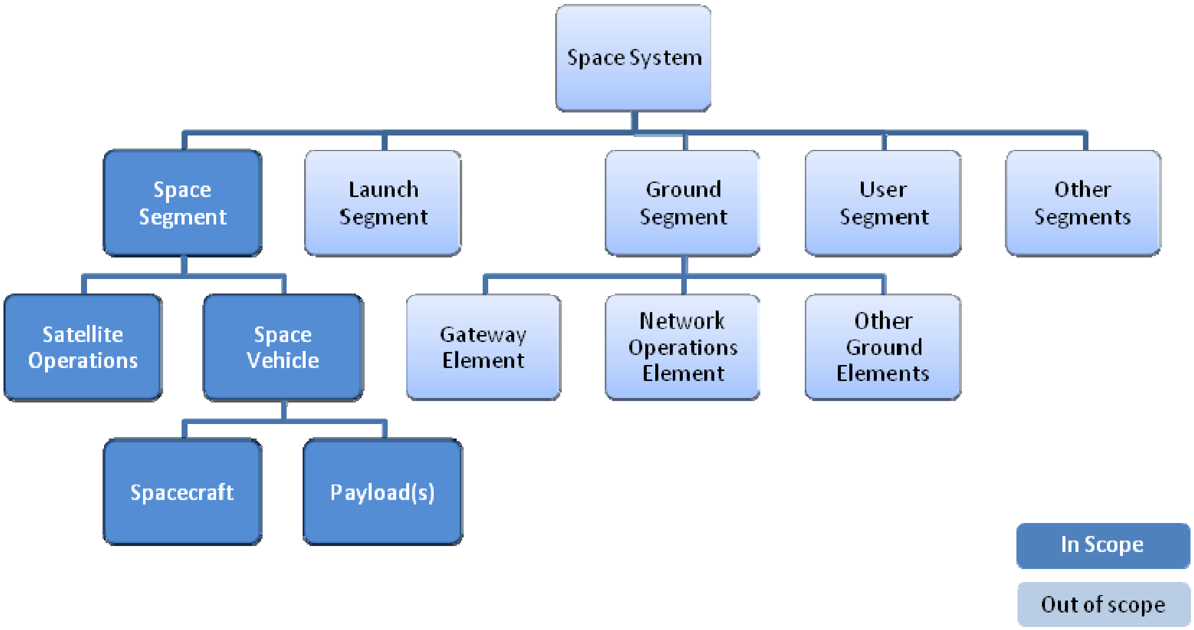


Figure 1. Scope of space system software readiness assessment.

The scope of the SRA as defined in this document includes the software portion of firmware but not the hardware portion. It does not include the software prepared as part of Application-Specific Integrated Circuit (ASIC) or Field Programmable Gate Array (FPGA) development that is used to drive the foundry equipment to produce the device.

1.3 Independence in a Software Readiness Assessment

The SRA should be chartered by a government or contractor person who has authority to act on the assessment team's recommendations. The assessment team may consist of personnel outside of or within the program, as long as they are independent of the personnel responsible for the products under development. The software readiness assessment may be conducted as a software-only risk assessment, as part of a program independent review, or in preparation for a program milestone or gated event.

1.4 SRA Process Overview

The SRA team evaluates the software products, processes, and resources, and determines the degree to which the assessment criteria are met, with rationale as to why the SRA team has made these judgments. The SRA team synthesizes these results into a risk assessment for proceeding to the next phase of software development or the identified upcoming event. The SRA results in a summary risk assessment for continued development. There are three choices for this summary risk assessment:

1. **Low risk.** The assessment team has identified no or minor deficiencies. Corrections can be made while software development continues as planned.

2. **Moderate risk.** The assessment team has identified at least one deficiency within the scope of the software development team for which corrective action is recommended before continuation to the next phase or milestone. There may also be deficiencies in the low risk category.

3. **High risk.** The assessment team has identified at least one deficiency that could lead to program failure. There may also be deficiencies in the low and moderate risk categories. Scope, schedule, resources, or technology maturity are misaligned. Program corrective action for the high risks is required before continuation to the next phase or milestone.

The results of the SRA are documented in a briefing that identifies the key risk areas and provides the summary risk assessment. The SRA process is described in detail in Section 2.2 below.

1.5 Assessment Points in the Development Life Cycle

SRAs are positioned at predefined points in the development life cycle where the readiness of the software for the next phase of development, or for an upcoming designated event, needs to be understood, along with the associated risk of proceeding. While there are numerous points in the development life cycle where such an assessment may be performed, this document addresses only two. The first assessment point is the Software Architecture Readiness (SAR). The SAR is conducted at the completion of software development planning and software architecture definition. The second assessment point is the Build Turnover Readiness (BTR). The BTR is conducted when a software build is turned over to an external organization for integration and test into the larger system.

While this version of the document contains criteria for only two assessment points, the authors strongly recommend that SRAs be performed as a routine way of doing business throughout the development life cycle. In particular, it is recommended that SRAs be performed early and often in order to identify the risks and issues in the software development as they occur, so that early corrective action is possible. Section 2.1 discusses in more detail how these assessments are positioned in different software life cycle models.

1.6 Effective Use of This Document

This document provides guidance for the planning, preparation, conduct, and completion of SRAs. It is intended to be used by development contractor teams, government teams, or combined development contractor-government teams. Government teams may include Federally Funded Research and Development Center (FFRDC), Systems Engineering and Integration (SE&I), or Systems Engineering and Technical Assistance (SETA) personnel in addition to, or instead of, government personnel.

Software expertise is required to properly apply the assessment criteria contained in this document. The criteria assume a depth of understanding in all software engineering activities, products, and processes as well as a depth of understanding of the space segment software domain and its associated issues and risks. The SRA team members must also have sufficient experience in software development and management, so that they are capable of identifying risks and properly assessing their impact and likelihood. The criteria should not be used as a checklist by personnel who do not have this level of software expertise.

When multiple SRAs are performed on a program, as recommended in Section 1.5 above, the ideal situation would be to have continuity of personnel across the assessment teams. While this is not always possible, a criterion for selecting SRA team members should be their participation on prior SRAs for the program.

This document is not written as a standard, and thus should not be used on contracts as a compliance document. If the government intends to perform SRAs or expects the development contractor to perform SRAs, tasking for this should be included in the statement of work, as well as requirements for government program office access to software products and processes. In addition, this TOR should be cited as a reference document for that tasking. Care should be taken to ensure that the number and scope of the contractually required SRAs are consistent with the scope, criticality, and integrity level of the software under development. Tailoring of the SRA is discussed in Section 2.3 below.

While this document specifically describes criteria for a software readiness assessment of a program by an independent team, the criteria may be used in other ways. A gap analysis can be performed between the organizational or program software processes, standards and procedures and the criteria in this document. The organizational or program software processes, standards, and procedures can then be updated where they are deficient against the criteria. Another possible use of the criteria is a self-assessment by the software development team so that the team can fix identified deficiencies before the next phase of development or program event or in preparation for an upcoming SRA.

1.7 Document Contents

Section 1 of this document provides introductory material, and Section 2 describes how to apply SRAs. Section 3 contains the assessment criteria for the products, processes, and resources. Appendices provide supporting information, such as acronyms and abbreviations, a glossary, and a reference list.

The term “should” is used throughout this document to describe activities that the authors recommend to be performed for an SRA. Since this is a guidance document, the word “shall” is not used.

2. Application of Software Readiness Assessments

2.1 Positioning of Assessments in the Software Development Life Cycle

This section describes the positioning of software readiness assessments in the software development life cycle and the associated position of these assessments in the system development life cycle. This includes a description of the objectives of the software readiness assessment at each point.

2.1.1 Software Readiness Assessment Points

The two software readiness assessment points covered in this document are:

1. Software Architecture Readiness (SAR), an early software life cycle point, occurs at the completion of software development planning and software architecture definition. The objective of the SAR assessment is to identify risks due to incomplete or immature software planning, software architecture and software requirements. It is essential to identify these risks early in development before significant resources are expended.

2. Build Turnover Readiness (BTR), a late software life cycle point, occurs:
   a. For onboard software, prior to loading the software into the flight hardware;
   b. For ground software, prior to installing the software into the operational ground facility or starting rehearsals, whichever comes first.

The objective of the BTR is to identify risks due to incorrect or immature software executables, user documentation and transition planning. Open discrepancy reports, problem reports or change requests should be used to assess the correctness and maturity of the software.

These readiness assessment points are the minimum recommended for a program. Performing assessments early and often is preferred. These assessments may be part of an overall independent and milestone review plan, or can be conducted separately. Each assessment should consider the results, actions and identified risks of prior assessments and reviews as an input. The results of the most recent software assessment should be addressed in subsequent milestone reviews.

2.1.2 Software Development Life Cycle Models

The software industry uses a variety of development life cycle models, each of which can be effective under specific circumstances. This document does not advocate a specific software development life cycle model, and there is no assumption as to the order in which software development activities are conducted. To provide perspective in applying software readiness assessments, this section addresses four commonly used software life cycle models: waterfall, incremental, iterative Unified Process, and agile. Note that each life cycle model has its own nomenclature for development activities and deliverables, and the assessors should be skilled in applying the assessment criteria to the chosen life cycle model. With the exception of this section, this document is written in a life cycle model-neutral fashion focused on processes, products, and resources and uses nomenclature aligned with the *Software Development Standard For Space Systems* (TOR-2004-(3909)-3537B).

Software development life cycle models are not the same as system development life cycle models. Synchronizing software development milestones with system milestones can be difficult. As an example, the traditional system development life cycle model is a “waterfall” life cycle model, where each activity is performed once. In contrast, there are a number of software lifecycle models where the activities repeat.

The following sections describe how software readiness assessments and system events may be related for the four selected software development life cycle models.

2.1.2.1 Waterfall Software Development Life Cycle Model

Figure 2 shows the waterfall software development life cycle model and its relationship to the system development life cycle model. In the waterfall life cycle, software development starts after system requirements have been defined and allocated to software. All software development is complete and software is qualification tested before a single turnover to system integration and test.

In the waterfall life cycle model, SAR should be performed early in the system and software design phase. Ideally, it is performed at the point when the software architectural (or high level) design is complete and prior to completing detailed software design. Architectural completeness with respect to the program's required quality attributes (reliability, dependability, maintainability, etc.) is the critical readiness criteria. For this reason, architectural design should complete well before program PDR and should guide the allocation of detailed software requirements.

Although Figure 2 shows BTR occurring at the end of software qualification testing, this is the ideal. In many cases, an early release of the software is made to system test or to the next level of integration prior to completion of software qualification testing. In this case, BTR should be performed in conjunction with the software release to system test or the next level of integration. A BTR should also be performed for in-process releases of software to external integration or test facilities.

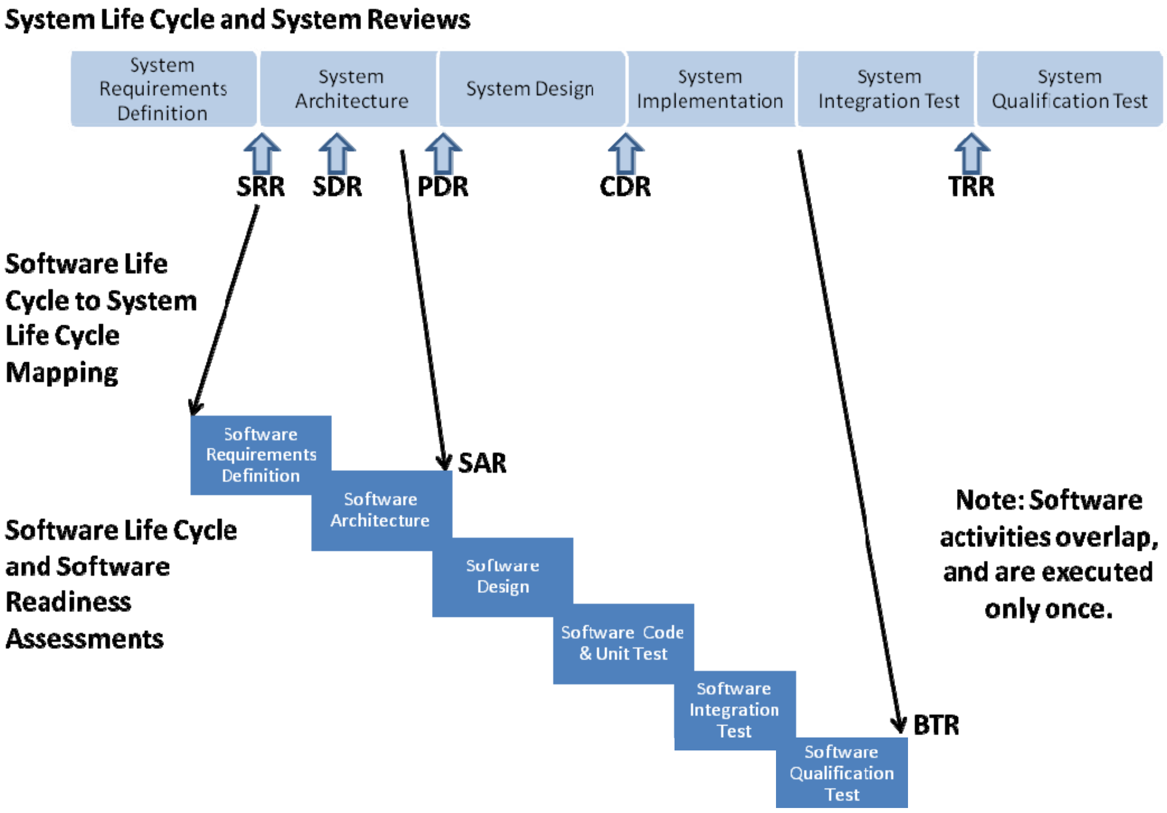

System Life Cycle and System Reviews
System Requirements Definition
System Architecture
System Design
System Implementation
System Integration Test
System Qualification Test
SRR
SDR
PDR
CDR
TRR
Software Life Cycle to System Life Cycle Mapping
Software Requirements Definition
SAR
Software Architecture
Software Design
Software Code & Unit Test
Software Integration Test
Software Qualification Test
BTR
Software Life Cycle and Software Readiness Assessments
Note: Software activities overlap, and are executed only once.

Figure 2. Waterfall life cycle model.

2.1.2.2 Incremental Software Development Life Cycle Model

Figure 3 shows the incremental software development life cycle model and its relationship to the system development life cycle model. In the incremental life cycle model, software development begins after the system requirements have been defined and allocated to software. The software requirements and architecture are defined up front, to the extent possible, based on the allocated system requirements. The software requirements are then partitioned into increments for implementation. Each increment implements the software satisfying its collection of software requirements through a sequence of the following software development activities: design, code and unit testing, and integration and testing.

The first increment focuses on software architecture decisions. The software architecture is validated to ensure the software requirements will be met, including the quality attributes (reliability, dependability, maintainability, etc.), and will accommodate all of the remaining increments without substantial rework.

The last increment ends with software qualification testing of the entire software product. Earlier increments may also include software qualification testing, especially if that increment is provided to an external organization for integration into the next level or for test bed or test facility checkout. Each increment after the first is integrated with the preceding increment, and the increments usually overlap in time.

In the incremental life cycle model, SAR should be performed early in the system design phase after the principal software architecture decisions are complete. Software architectural completeness with respect to the program's required quality attributes is the critical readiness criteria. The software architecture must also be able to serve as the framework for implementing the remaining increments and satisfying all of the software requirements. For these reasons, the SAR is generally held before the program PDR.

In the incremental life cycle model, software qualification testing is performed on the final increment, and BTR should be performed after the completion of the software qualification testing and before the software is released to the next higher level of integration. Build Turnover Readiness (BTR) assessments should also be held for any early increments that are released to an external organization.

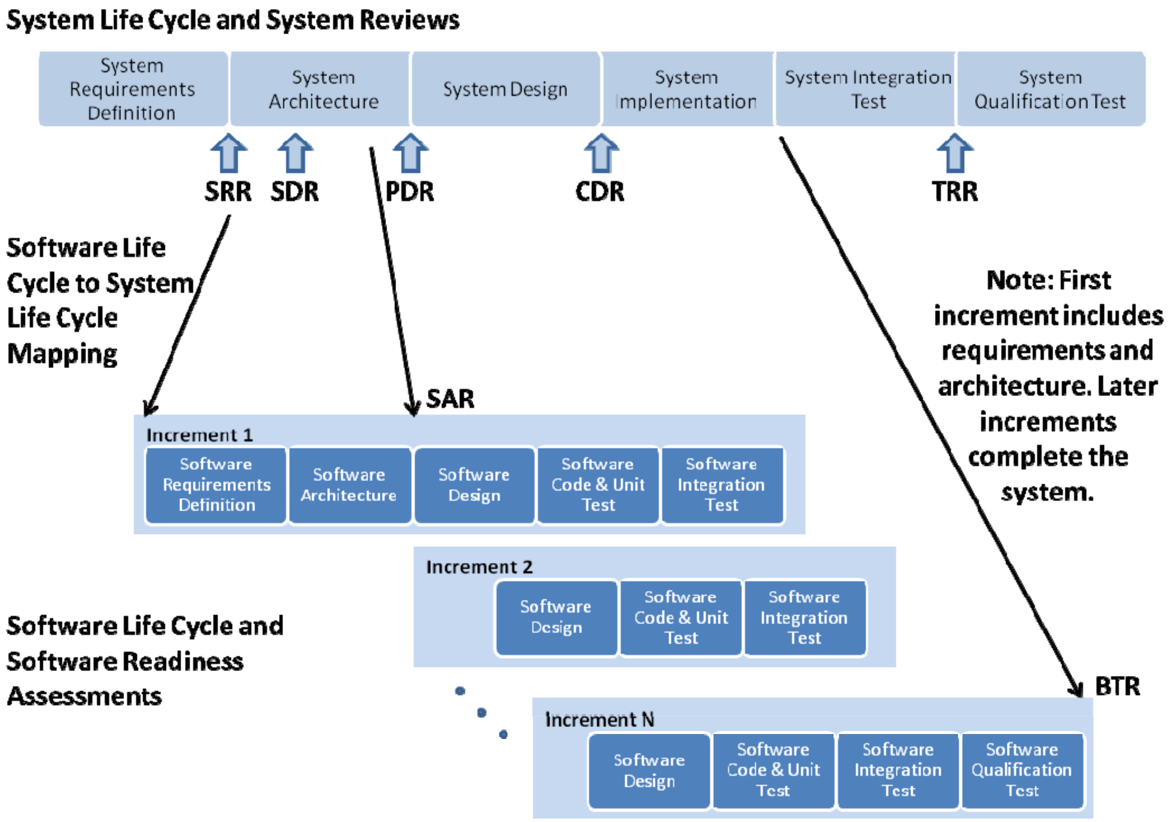

System Life Cycle and System Reviews
System Requirements Definition
System Architecture
System Design
System Implementation
System Integration Test
System Qualification Test
SRR
SDR
PDR
CDR
TRR
Software Life Cycle to System Life Cycle Mapping
SAR
Note: First increment includes requirements and architecture. Later increments complete the system.
Increment 1
Software Requirements Definition
Software Architecture
Software Design
Software Code & Unit Test
Software Integration Test
Increment 2
Software Design
Software Code & Unit Test
Software Integration Test
Software Life Cycle and Software Readiness Assessments
BTR
Increment N
Software Design
Software Code & Unit Test
Software Integration Test
Software Qualification Test

Figure 3. Incremental software development life cycle model.

2.1.2.3 Iterative Unified Process Software Development Life Cycle Model

Figure 4 shows an iterative life cycle model used in the Unified Process and its relationship to the system development life cycle model. The Unified Process includes four phases: inception, elaboration, construction, and transition. Within the phases, the software is partitioned into iterations, shown by the small rectangles in the figure. Each iteration goes through the entire software development life cycle of requirements definition, architecture definition, design, code and unit testing, and integration and testing.

The objective of the software iterations performed in the inception phase is to provide data for system trades and to perform sufficient risk reduction to ensure the viability of proceeding with development. In the traditional system development waterfall life cycle, the inception phase is completed prior to authorization to proceed (e.g., prior to the beginning of the contract to develop the system).

The objective of the software iterations in the elaboration phase is to develop an executable version of the software architecture, derive and analyze all architecturally significant requirements, and mitigate development risks. Typically, the elaboration phase ends prior to program PDR. In this life cycle model, SAR should be performed during the elaboration phase, prior to program PDR.

The objective of the software iterations in the construction phase is to develop the software to satisfy the allocated system requirements. Some iterations during the construction phase may be released to an external organization for integration into the next level. Software qualification test is often not performed until the final construction iteration is complete. In the Unified Process, ideally BTR should be conducted iteratively. While it is not necessary to perform a BTR for each development iteration, it is essential that a BTR be performed at each incremental release to system test or the next level of integration. At a minimum, BTR should be performed at the end of the last construction iteration, prior to transitioning to operations, at the start of the transition phase.

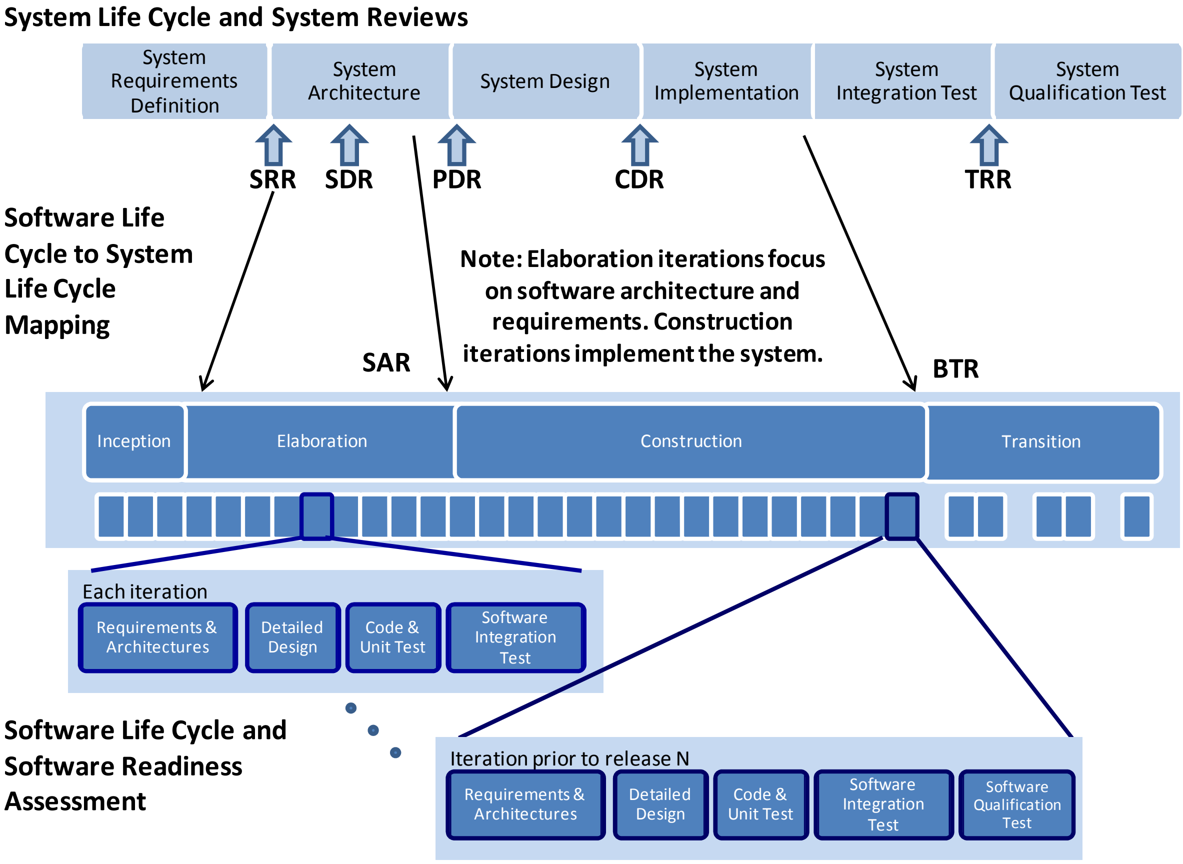

System Life Cycle and System Reviews
System Requirements Definition
System Architecture
System Design
System Implementation
System Integration Test
System Qualification Test
SRR
SDR
PDR
CDR
TRR
Software Life Cycle to System Life Cycle Mapping
Note: Elaboration iterations focus on software architecture and requirements. Construction iterations implement the system.
SAR
BTR
Inception
Elaboration
Construction
Transition
Each iteration
Requirements & Architectures
Detailed Design
Code & Unit Test
Software Integration Test
Software Life Cycle and Software Readiness Assessment
Iteration prior to release N
Requirements & Architectures
Detailed Design
Code & Unit Test
Software Integration Test
Software Qualification Test

Figure 4. Iterative unified process software development life cycle model.

2.1.2.4 Agile Software Development Life Cycle Model

In the agile life cycle model, shown in Figure 5, software development is performed in a series of cycles (shown in the figure as small rectangles) for initiation, development, and production. Initiation (cycle 0) focuses on team and development environment readiness. The software development is partitioned into cycles that result in releases. There are both development and production releases, where production releases occur during operations and maintenance. Each cycle includes the software development activities of requirements definition, architecture, design, code and unit test, and integration and test. The cycles are generally very short (weeks) in duration. Some development releases may be turned over to an external organization for integration into the next level. Software qualification test is often not performed until the final development release is complete. Examples of agile software development methods used in the agile life cycle model include eXtreme Programming and Scrum.

In the agile software development life cycle model, SAR should be performed within the first few development cycles as the primary architectural features of the software are completed. Agile life cycles often emphasize iterative delivery of business value rather than iterative reduction of program risk. For this reason, architectural agility and adaptability may be more critical than architectural completeness, and the SAR product criteria should be tailored to reflect this. If significant architectural refactoring is performed at any point in the program, readiness against the SAR criteria should be re-assessed.

In the agile software development life cycle model, it is not necessary to perform a BTR for each development release. It is, however, essential that a BTR be performed for each release going to production.

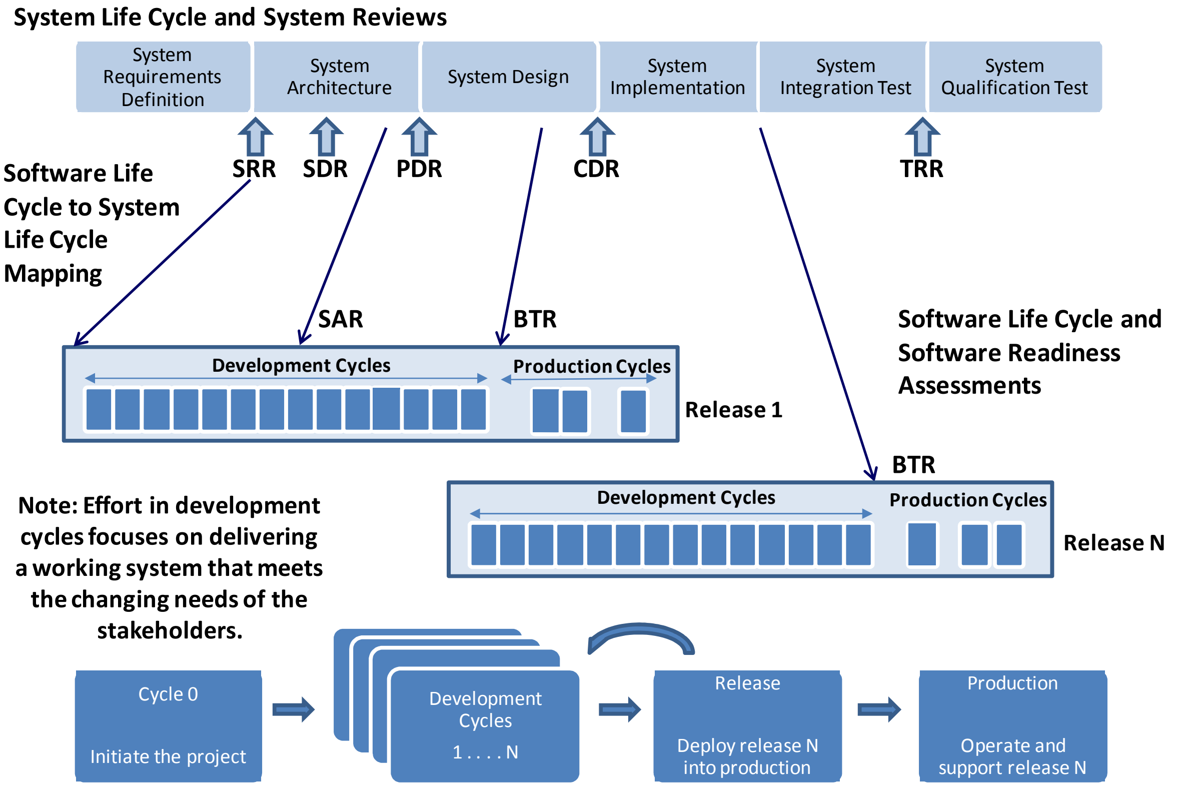

System Life Cycle and System Reviews
System Requirements Definition
System Architecture
System Design
System Implementation
System Integration Test
System Qualification Test
SRR
SDR
PDR
CDR
TRR
Software Life Cycle to System Life Cycle Mapping
SAR
BTR
Software Life Cycle and Software Readiness Assessments
Development Cycles
Production Cycles
Release 1
BTR
Development Cycles
Production Cycles
Release N
Note: Effort in development cycles focuses on delivering a working system that meets the changing needs of the stakeholders.
Cycle 0
Initiate the project
Development Cycles
1 . . . . N
Release
Deploy release N into production
Production
Operate and support release N

Figure 5. Agile software development life cycle model.

2.1.3 Potential Software Readiness Assessment Points

Table 1 provides an example list of potential software readiness assessment points for a program. The two software readiness assessment points covered in this document are highlighted in bold. This list should be tailored for the program's selected system and software life cycle models (see Section 142.3).

Table 1. Potential Software Readiness Assessment Points

SW Readiness Assessment Points
1. Software Initiation Readiness (IBR Timeframe)
2. Software Planning and Process Readiness (Readiness for use of plans and processes for software development)
3. Software Readiness for SRR
4. Software Readiness for SFR (or SDR)
5. Software Architecture Readiness (SAR)
6. Software Readiness for PDR
7. Software Readiness for CDR
8. Software Build Readiness (for each build) for: a. Software Build Baseline Configuration Readiness (detailed build planning review) b. Software Build Design Readiness c. Software Build Test Readiness d. Software Build Exit Readiness
9. Software Readiness for TRR
10. Build Turnover Readiness (BTR)
11. Software Readiness for Pre-Ship Review (PSR)
12. Software Readiness for Mission Readiness Review (MRR)
13. Software Readiness for Flight Readiness Review (FRR)
14. Software Readiness for Transition to Operations (may include Initial Checkout Review [ICR])
15. Software Readiness for Transition to Maintenance

BTR	Build Turnover Readiness	PSR	Pre-Ship Review
CDR	Critical Design Review	SAR	Software Architecture Readiness
FRR	Flight Readiness Review	SDR	System Design Review
I&T	Integration and Test	SFR	System Functional Review
IBR	Integrated Baseline Review	SRR	System Requirements Review
ICR	Initial Checkout Review	SW	Software
MRR	Mission Readiness Review	TRR	Test Readiness Review
PDR	Preliminary Design Review		

2.2 Planning and Performing the Assessment

A software readiness assessment for a major space system involves many stakeholders: the readiness assessment team, the contractor's software and systems engineers, government software and systems engineers, and management. In order to produce an effective and efficient assessment, good planning is necessary. The sequence of activities involved in planning and performing a software readiness assessment is shown in Figure 6, and described in the subsequent paragraphs of this section. Depending on the requirements, scope, and constraints of the software development, the steps of this process may be tailored, as discussed in Section 2.3 below.

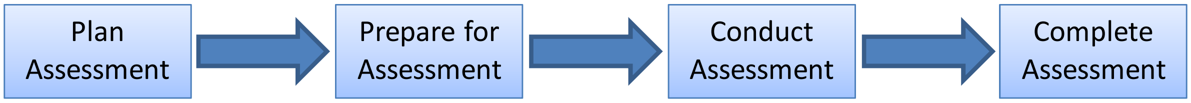


Figure 6. Steps in planning and performing a software readiness assessment.

2.2.1 Plan the Assessment

As discussed in Section 1, a software readiness assessment is chartered by a contractor or government person who has the authority to act on the assessment team's recommendations. The person who charters the readiness assessment is called the sponsor. The sponsor selects the readiness assessment team lead. The team lead should be an experienced software developer or manager with experience leading software assessments and with the appropriate domain expertise for the software to be assessed. The team lead, with the sponsor's approval, selects the remaining team members. The team members should be experienced software developers or managers with the appropriate domain expertise for the software to be assessed. The assessment team may consist of personnel outside of or within the program, as long as they are independent of the personnel responsible for the development of the products under review. The software readiness assessment may be conducted as a software-only risk assessment, as part of a program independent review, or in preparation for a program milestone or gated event.

The assessment team tailors the assessment scope, objectives, and criteria to be consistent with the requirements of the contract, the magnitude of the software effort in the program, the program phase, the software development life cycle model, specific program terminology, and any other constraints imposed upon the assessment. Planning should include review of action items from previous software assessments and other gated events to determine if the actions have been closed, or that they continue to be worked and have a valid closure plan. The results of this review should be used in the tailoring of the criteria for the current assessment.

The output of the planning process is a documented assessment plan that includes:

- The tailored assessment criteria
- The resources required for the assessment, including:
  - Training required for the assessment team
  - Orientation for the software development team
  - Support from the software development team
  - Tools for use in assessment (e.g., spreadsheets for criteria and evidence tracking, risk, cost, schedule analysis tools)
- The estimated cost, schedule, and duration of the assessment
- The logistics of the assessment, including access to facilities and the electronic repositories or tools containing the objective evidence
- The disposition of the assessment team's materials at the conclusion of the assessment
- Whether the assessment team will provide risk mitigation recommendations as part of the findings
- Any other special considerations

The assessment plan is approved by the sponsor and any other required stakeholders to obtain commitment to the plan. After the plan is approved, preparation for the assessment can begin.

2.2.2 Prepare for the Assessment

The success of a readiness assessment depends on the quality and preparation of the readiness assessment team. It is the responsibility of the assessment team leader to select qualified members of the team and to ensure that they receive any necessary training to participate on the team. Training may include presentations by the contractor to familiarize the assessment team with the program. Assessment team members may need to be trained in the tools that the team will use in the assessment. To be effective, team members should be committed to the assessment from the planning stage through conduct and completion.

The assessment team provides orientation to the software development team to communicate the scope of the assessment. The assessment team works closely with the program's software development team to obtain access to the objective evidence that the assessment team requires.

A major component of performing a software readiness assessment is the evaluation of objective evidence that activities are being performed as required by the program's statement of work and the contractor's organizational processes. Sources of objective evidence can include electronic repositories, presentations, documents, and interviews. Objective evidence includes artifacts that are the required output of the processes executed. For example, the objective evidence that software requirements are developed is the existence of software requirements in a requirements management tool, along with bi-directional traceability and identified verification methods. The objective evidence that peer reviews are conducted is the documented minutes from the peer reviews showing that the required personnel were in attendance, adequate preparation was performed, reasonable findings were documented, and additionally, open items from the peer review are tracked through successful closure. The criteria in Section 0 describe the quality attributes of the objective evidence for processes, products, and resources.

Preparation for a software readiness assessment largely consists of identifying the sources of the objective evidence and obtaining access to those sources. The readiness assessment team may need to work with the software developers to explain what constitutes objective evidence for the criteria. The readiness team may choose to use a tool, such as a spreadsheet, to catalogue the evidence used in the assessment. This evidence may then be mapped to the tailored assessment criteria to demonstrate whether the criteria have been satisfied.

Preparation should include a presentation by the software development team to the software readiness assessment team and other stakeholders. This is an opportunity for the software development team to explain the purpose of the software under development, the life cycle model and processes in use on the program, the current status of the development, and why they believe that software development is ready to move to the next phase. It is also an opportunity for the software readiness assessment team to ask clarifying questions about the objective evidence that they are seeking.

The output of the preparation process is a qualified and trained assessment team, a set of tools (e.g., spreadsheets with tailored assessment criteria) to facilitate the assessment, and the identification of and access to the objective evidence needed for the assessment. The team has access to all the objective evidence required for the assessment, and the program's software team is prepared to discuss the evidence with the assessment team.

2.2.3 Conduct the Assessment

The assessment team evaluates the objective evidence to determine compliance with required product, process, and resource criteria, and notes the degree to which the criteria are met. If evidence is missing for an activity or if the evidence shows poor compliance with the criteria, the team documents the gap.

The assessment team should leverage the results of peer reviews and other product evaluations. Since peer reviews have been proven to be one of the most important factors in the quality of software products, it is important that the assessment team pay particular attention to the quality and effectiveness of the peer review process. The results of all product evaluations should be a key factor in the assessment.

After performing an initial review and analysis of the evidence, the assessment team should review their findings with the software team. The software team may be able to provide additional evidence that was overlooked the first time. This process may iterate several times.

The determination of how well the objective evidence supports the criteria is based on the experienced judgment of each member of the assessment team and the consensus of the entire team, for each criterion. Rating worksheets and automated support tools can facilitate the team's decision-making process by presenting necessary data in a concise, well-organized manner. The team lead facilitates the process of achieving team consensus and the team's findings are documented.

Where gaps exist between the program's performance and the criteria, the team makes an assessment of the risks that the gap produces. Since the criteria represent best industry practices, deviations from the criteria suggest risks to the program. The assessment team synthesizes the deficiencies for individual criteria into higher level findings and for each finding, determines the associated issues and risks. The synthesis occurs both within and across products, processes, and resources. Based on the risks, the readiness assessment team makes a summary assessment of the risk of proceeding, as defined in Section 1.4.

If requested by the sponsor, the risk assessment includes recommendations for how to handle the risks. Some risks may require mitigation or contingency plans with associated cost and schedule implications. The assessment team documents the risks and risk handling recommendations.

The output of the assessment is the documented findings, risks, and issues, and the summary risk assessment.

2.2.4 Complete the Assessment

The readiness assessment team communicates its findings, risks, and issues, and summary risk assessment to the sponsor and other stakeholders. The sponsor determines any follow-up actions for the readiness assessment team.

The readiness assessment team archives or disposes of the team's working papers, as agreed to in the plan. The readiness assessment team documents any lessons learned for use in future readiness assessments.

2.3 Tailoring

This document is meant to be tailored when applied to ensure that only necessary and cost-effective assessments are conducted. The tailoring should be based on the life cycle, nomenclature, and processes of the program being assessed. In each application, the assessment should be tailored to the specific needs of a particular program or program phase and life cycle model used. Tailoring is the responsibility of the assessment team, under the approval of the assessment sponsor. The sponsor may provide suggested tailoring to assist in identifying the areas considered the highest risk for a particular contract. Both the assessment process and the criteria of the assessment may be tailored.

The software readiness assessment process can be tailored to address a variety of attributes, such as program size, software criticality, and resource availability. Care should be taken to reduce the formality of, or eliminate tasks that add unnecessary costs or do not add value to the process or the product. For instance:

- The size of the assessment team should be scaled to the size of the program. For a very small program, the assessment team might consist of only one person.

- Organizing a small internal assessment team can be as simple as sending an email request. An assessment team that is comprised of government, prime, and supplier personnel may require more formality to organize.

- Evidence may take different formats (e.g., paper vs. electronic) depending on the assessment team's access to and familiarity with the repositories containing the artifacts. As an example, if the assessment team has access to all the repositories, the evidence might exist only in electronic format, whereas a team consisting of personnel from multiple organizations may operate more effectively with some evidence being provided in paper format.

- The output of an assessment of non-mission critical software may be provided in an informal meeting with the sponsor. An assessment for mission critical software, however, may require formal outputs in the form of documented presentations.

The criteria for the assessment may also be tailored. The ultimate goal of tailoring is not to dilute the criteria, but rather to focus the assessment in such a way as to obtain the greatest value from it, consistent with the level of risk for the software. Tailoring of the criteria takes the form of:

- Alteration of criteria to reflect program-familiar terminology or artifacts, such as combining the products (outputs) listed in the criteria to reflect the program's required deliverables.

- Addition of criteria to provide special focus consistent with program requirements.

- Deletion of criteria when the activity is not applicable to the contracted program scope.

- Alteration of the assessment points at which the tailored criteria are applied.

3. Process, Product, and Resource Perspectives

As already described in Section 2, software development may follow many different life cycle models, software development methodologies and programming paradigms. In this section we present three different perspectives or viewpoints on space segment software development: product, process, and resource.

Each of these perspectives is intended to illuminate best software development practices and illustrate the level of maturity expected at each assessment point (e.g., SAR, BTR). The assessment criteria and assessment points are intended to span all software life cycle models, but there may be cases where a criterion does not apply. For example, some criteria in the early life cycle assessment (SAR) may apply to an iterative or agile development but not to a waterfall development. These types of conditional situations are clarified by the use of terminology like “for any code developed by SAR” or “for any test procedures developed by SAR.” In these cases the criteria should be applied only to the work that has been completed at that point. At BTR, referring to the SAR criteria may provide additional context that will clarify the intent of the BTR criteria.

A number of other conventions have also been used in developing the specific wording of the assessment criteria. The intent of the criteria is to establish “what” should to be done, but not necessarily “how” it should be done. Criteria that refer to documents or plans (e.g., software development plan or software architecture description) may be satisfied either by a document or by data in a tool or electronic repository. General terms such as “assessment” or “evaluation” are used rather than “appraisal” to avoid any connotation of CMMI or SCAMPI. “Software product evaluation” is used as a general term for product reviews which can take different forms. “Peer review” refers to a more specific type of product review with specific expectations. The glossary in Appendix A defines and clarifies many of the terms used in the tables.

3.1 Assessment Criteria for the Product Perspective

Table 2 contains the criteria for SRAs for the product perspective. It contains four columns. The first column identifies the software product being assessed. The second column identifies an aspect of the product to be assessed. The third column provides the assessment criteria for the Software Architecture Readiness (SAR) for that product and area. The last column provides the assessment criteria for the Build Turnover Readiness (BTR) for that product and area.

Note that some software plans (e.g., software development plan, software configuration management plan) are in Table 3, because they are the output of the planning process. Test plans, however, are in Table 2, along with the other testing products.

3.2 Assessment Criteria for the Process Perspective

Table 3 contains the criteria for SRAs for the process perspective. It contains four columns. The first column identifies the software process being assessed. The second column identifies an aspect of the process to be assessed. The third column provides the assessment criteria for the Software Architecture Readiness (SAR) for that process and area. The last column provides the assessment criteria for the Build Turnover Readiness (BTR) for that process and area.

3.3 Assessment Criteria for the Resource Perspective

Table 4 contains the criteria for SRAs for the resource perspective. It contains four columns. The first column identifies the software resource being assessed. The second column identifies an aspect of the resource to be assessed. The third column provides the assessment criteria for the Software Architecture Readiness (SAR) for that resource and area. The last column provides the assessment criteria for the Build Turnover Readiness (BTR) for that resource and area.

Table 2. Assessment Criteria for the Product Perspective

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
System Requirements	Specification Tree	The system specification tree identifies all software specifications, and is consistent with the system architecture. The system specification tree is complete, baselined, stable, and released.	The system specification tree is complete, current, baselined, stable, and released.
System Requirements	Requirements Allocated to Software	System requirements allocated to software (including software Key Performance Parameters/Key Performance Measures (KPPs/KPMs); computer resource margins; and functional, performance, and specialty engineering requirements) are complete, baselined, stable, and released. Analysis or modeling substantiates the allocation of quantitative performance requirements to software, hardware, and people.	System requirements allocated to software are complete, current, baselined, stable, and released. The risks associated with any late or unincorporated changes have been assessed, documented, and communicated; an acceptable risk handling plan is in place.
System Requirements	Requirements Allocation Matrix	The bi-directional mapping between the software items in the system architecture and the system requirements allocated to software is complete, baselined, stable, and released.	The bi-directional mapping between the software items in the system architecture and the system requirements allocated to software is complete, current, baselined, stable, and released.
System Requirements	Specialty Engineering	System specialty engineering requirements (including reliability, maintainability, and availability; safety; information assurance; and human systems engineering) are defined, appropriately allocated to software, and complete, baselined, stable, and released. Analysis or modeling substantiates the allocation of quantitative specialty engineering requirements to software, hardware, and people.	System specialty engineering requirements allocated to software are complete, current, baselined, stable, and released.
System Requirements	Data Management	System requirements for management of on-board and ground data are defined, appropriately allocated to software, and complete, baselined, stable, and released.	System requirements for management of on-board and ground data are complete, current, baselined, stable, and released.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
System Requirements	Fault Management	System requirements for management of on-board and ground faults, including safe mode, are defined, appropriately allocated to software, and complete, baselined, stable, and released.	System requirements for management of on-board and ground faults are complete, current, baselined, stable, and released.
System Requirements	Reprogramability	System requirements for reprogrammability are defined, appropriately allocated to software, and complete, baselined, stable, and released.	System requirements for reprogrammability are complete, current, baselined, stable, and released.
System Interface Requirements	System Interfaces	The space-to-ground, space-to-space, and ground-to-ground logical interface definitions are complete, baselined, stable, and released.	The space-to-ground, space-to-space, and ground-to-ground physical interface definitions are complete, current, baselined, stable, and released. The risks associated with any late changes have been assessed, documented, and communicated; an acceptable risk handling plan is in place.
System Architecture	Fault Management	System architecture addresses overall fault management and the Operations Concept (OpsCon) is sufficiently defined to allocate design to software architecture. Fault management is identified as a critical component of the architecture requiring the highest integrity level.	Fault management and recovery has been validated, verified and stress tested in an operationally relevant environment. The highest integrity level analysis has been completed; static analyses of state variable transition models and stress testing of dynamic timing risks have been successful. As-built design, assumptions, rationale, and operational environment and performance limits have been documented. All risks identified in criticality analyses have been mitigated.
System Architecture	Traceability	System architecture components are bi-directionally mapped to system, subsystem, or element requirements. The bi-directional traceability is complete, baselined, stable, and controlled.	The system architecture to system, subsystem or element requirements bi-directional traceability is complete, current, baselined, stable, and controlled.
System Architecture	Completeness	System architecture trade studies are complete and documented. System architecture decisions have been finalized. The system architecture description is complete, baselined, stable, and released, and addresses hardware, software, and data or databases.	The system architecture description is complete, current, baselined, stable, and released. The risks associated with any late changes have been assessed, documented, and communicated; an acceptable risk handling plan is in place.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
System Operations Concept	Completeness	The system OpsCon is established and is sufficiently mature to enable the software team to validate the software architecture. All operations are identified and described, including nominal and off-nominal scenarios. There are no open issues or inconsistencies. High risk areas are described to a very low level. The system OpsCon is complete, baselined, stable, and released.	The system OpsCon is completed, current, baselined, stable, and released. Late changes may indicate there are issues with the final system/software implementation or its intended use was not adequately described in the initial architecture phase.
System Scenarios	Completeness	System scenarios affecting the software requirements or software architecture are complete, baselined, stable, and controlled. System scenarios (e.g., use cases and threads) implement the system operations concept.	System scenarios that affect the software requirements or software architecture are complete, current, baselined, stable, and controlled.
System Scenarios	Test Like You Fly	System scenarios including "Test Like You Fly" (TLYF) and "Day in the Life" (DITL) are defined, baselines, stable, and controlled.	System scenarios that affect the software requirements or software architecture are complete, current, baselined, stable, and controlled and specifically include TLYF and DITL scenarios.
Configuration Item List	Completeness	The software configuration item list is complete, baselines, stable, and controlled.	The software configuration item list is complete, current, baselined, stable, and controlled.
Algorithm Design Document	Completeness	The algorithm design document is complete, baselined, stable, and released. The algorithms are defined to an appropriate level of detail for software development.	The algorithm design document is complete, current, baselined, stable, and released.
Software Risk List	Completeness	Software technical risks have been identified, assessed and prioritized. Effective mitigation or contingency plans are in place and being executed. Program management is able to fund either the mitigation activity or the realized risk.	Technical risks associated with this build are retired. If any technical risks associated with this build have not been retired, the risks and their impacts have been communicated to the receiving organization.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Requirements	Completeness	In addition to the system requirements allocated to software, requirements analysis and derivation has been done to identify and capture the lower-level software requirements. Quality attributes and critical software requirements, including performance, security, interface and functional requirements have been defined.	The software requirements are complete, current, baselined, stable, and released. The risks associated with any waived, late or incomplete requirements have been assessed, documented and communicated; an acceptable risk handling plan is in place.
Software Requirements	Interfaces	Critical interface requirements are documented and allocated to software architecture components.	The software interface requirements are completed, current, baselined, stable, and released. The risks associated with any waived, late or incomplete interface requirements have been assessed, documented and communicated; an acceptable risk handling plan is in place.
Software Requirements	Standards	The software requirements include specification of all applicable standards (e.g., human-system interface, backwards compatibility, network protocols, product interoperability, middle-ware).	The software requirements include specification of all applicable standards.
Software Requirements	Fault Management	The software specifications include all applicable fault management and error handling requirements. For flight software this includes any autonomous responses to spacecraft hardware failures or anomalies.	The software requirements include specification of all applicable fault management and error handling requirements. No requirements for fault management or error handling have been waived.
Software Requirements	Traceability	Defined software requirements have bi-directional traceability with the parent requirements allocated to software and are complete, baselines, stable, and controlled.	Software requirements are consistent with the as-built system, have bi-directional traceability with the parent requirements allocated to software, and are complete, current, baselines, stable, and controlled.
Software Requirements	Completeness	Analysis has been performed that demonstrates that the software requirements satisfy their parent requirements and are consistent with documented operations concepts.	Software requirements completely satisfy all parent requirements allocated to software for this build and support the baselined operations concept.
Software Requirements	Verifiability	Analysis has been performed demonstrating the software requirements can be verified.	The as-built software requirements can be verified.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Requirements	Validation	Analysis has been performed validating the software requirements meet the intent of the user requirements and operations concept.	The as-built software requirements meet the intent of the user requirements and operations concept.
Software Architecture	Software Architecture Views	The software architecture principles (e.g., partitioning rationale, protection mechanisms, layering, integrated error/fault management) are clearly documented and consistently realized in the software architecture views. Use cases, state transition diagrams, etc. are used depending on the architectural modeling methods selected.	The software architecture views are complete, current, baselines, stable, and controlled. The risks associated with any late or unincorporated changes have been assessed, documented and communicated; an acceptable risk handling plan is in place.
Software Architecture	Software Architecture Description	The software architecture description contains rationale for selection of software architecture views and those views are consistent with system requirements and needs.	The software architecture description is complete, current, baselined, stable, and released.
Software Architecture	Software Architecture Components	Software architecture components are mapped to the system architecture components (e.g., via a deployment diagram).	The mapping of software architecture components to the system architecture components is complete, current, baselines, stable, and controlled.
Software Architecture	Consistency with Software Requirements	The software architecture is consistent with the software requirements and system needs.	The as-built software architecture is consistent with the software requirements and system needs.
Software Architecture	Design Standards and Practices	The software architecture conforms to the program's architectural design standards and practices.	The software architecture conforms to the program's architectural design standards and practices.
Software Architecture	Documentation	Software architecture trade studies are complete and documented. Software architecture decisions have been documented. Software architecture description is complete, baselined, stable, and released.	The software architecture description is complete, current, baselined, stable, and released. The software architecture description reflects the as-built system and is consistent with the as-built software. The risks associated with any late or unincorporated changes have been assessed, documented and communicated; an acceptable risk handling plan is in place.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Architecture	Algorithm Allocation	The algorithms defined in the algorithm design document are allocated to the software architecture components.	The algorithms allocated to this build are implemented in the software architecture components.
Software Architecture	Data Model	The software architecture defines the conceptual data model. The software architecture defines the logical data model for data in external interfaces. The proposed methods of operation (e.g., transactions, retrievals, updates) are consistent with the anticipated usage.	The physical data model is complete, current, baselined, stable, and released. The risks associated with any late or unincorporated changes have been assessed, documented and communicated; an acceptable risk handling plan is in place.
Software Architecture	Communication Patterns	The software architecture defines communication patterns (e.g., OSI seven layer model, CCSDS) for space-to-ground, ground-to-ground and space-to-space interfaces.	The communication pattern descriptions are complete, current, baselined, stable, and released. The risks associated with any late or unincorporated changes have been assessed, documented and communicated; an acceptable risk handling plan is in place.
Software Architecture	Hardware Compatibility	The software architecture is compatible with target hardware constraints and assumptions.	The software architecture is updated and documented in accordance with the actual target hardware.
Software Architecture	Testability	Architectural features required for all test phases are incorporated in the software architecture (e.g., insertion and observation of faults, observation of behavior and data results, hooks for drivers, simulators, test data analysis tools).	Software testability features are complete, current, baselined, stable, and released.
Software Architecture	NDI Selection	NDI (e.g., COTS, GOTS, open source, reuse) are selected based on trade study results and the computing resource requirements have been coordinated with the hardware selection team.	The selected NDI, as documented in the software architecture, is incorporated into the as-built software.
Software Architecture	NDI Analysis	NDI analysis (e.g., ability to meet requirements, reliability, maturity, availability, suitability for incorporating in architecture, COTS vendor viability) is complete.	NDI analysis is complete, current, baselines, stable, and controlled. The risks associated with any late or unincorporated changes have been assessed, documented and communicated; an acceptable risk handling plan is in place.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Algorithms	Completeness	Preliminary software algorithms are identified. Critical software algorithms have been prototyped in a target-like environment.	The software algorithms are implemented in accordance with the algorithm documentation.
Software Architecture	Traceability	Bi-directional traceability between software architecture components and software requirements is complete, baselines, stable, and controlled.	Bi-directional traceability between software architecture components and software requirements is complete, current, baselines, stable, and controlled.
Software Analyses	Resource Utilization	Initial resource utilization analyses (e.g., for processor throughput, memory, storage, and bandwidth) has been performed for all software items. Initial resource utilization budgets are allocated to software items and are consistent with margin requirements. If multiple software items share processor resources, allocations made total no more than 100% of the budget. Initial analyses demonstrate that the resource utilizations of each software item are within their budgets.	Final analysis results, including measured resource utilization of each software item in the build, demonstrate that the as-built software meets all allocated resource utilization budgets (e.g., processor throughput, memory, storage, and bandwidth).
Software Analyses	Database Timing	An initial database timing model is defined and demonstrates the ability of the database to meet the timing requirements.	The database timing model reflects the as-built software. The database timing model takes into account database transactions, updates and business rules under stressing scenarios in accordance with expected and off-nominal usage profiles. The database timing model has been evaluated using the as-built schema definitions and creation, backup and recovery scripts, and using validated and controlled command and telemetry data. Final analysis results demonstrate that the as-built software for this build satisfies its database timing requirements.
Software Analyses	Timing Requirements	An initial timing model is defined and demonstrates the ability of the software architecture to meet the timing requirements.	The timing model reflects the as-built software. Final analysis results demonstrate that the as-built software for this build satisfies its timing requirements.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Analyses	Software Assurance Cases	Assurance cases substantiate the ability of the software architecture to meet its critical allocated requirements. Modeling, prototyping, simulation or other analysis methods are used to demonstrate that the software architecture can meet critical requirements (e.g., performance, KPPs, margins, critical technology elements, safety, information assurance, dependability, reliability) without loss of software integrity even under stressing conditions. Functional and performance shortfalls are communicated and resolved.	Evidence substantiates the ability of the software to meet its critical allocated requirements (e.g., performance, KPPs, margins, critical technology elements, safety, information assurance, dependability, reliability) without loss of software integrity, even under stressing conditions is included in the assurance cases. The assurance cases are complete, current, baselines, stable, and controlled. Software assurance artifacts are maintained under CM to facilitate regression testing.
Software Analyses	Reliability	An initial software reliability model is defined demonstrating the feasibility of the software architecture to meet the reliability requirements. A plan exists for collecting appropriate data during testing to be used in a software reliability prediction model to demonstrate the as-built code meets the allocated reliability requirements.	Analysis demonstrates that the as-built code for this build meets the allocated reliability requirements.
Software Analyses	Criticality Analysis Report	A multi-level integrity scheme is established and a criticality analysis is completed mapping system requirements and architecture elements to the established integrity levels. At SAR, the mapping extends to the software and security requirements, interface requirements and software architecture elements or components. The analyses addresses the traceability, correctness, consistency, completeness, accuracy, readability and testability of the system and software requirements.	Criticality analysis has been updated from the prior report. Analyses of all identified critical elements have been updated for the as-built software requirements, interfaces, components, test plans and test procedures in accordance with their criticality level.
Software Design	Data Model	Initial implementation of the data model in the executable architecture is defined and validated for feasibility through prototyping, modeling, or simulation.	The implemented data model is robust, complete, current, baselined, stable, and released.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Design	Communication Patterns	Initial implementation of the communication patterns in the executable architecture is defined and validated for feasibility through prototyping, modeling, or simulation.	Implemented communication patterns are robust, complete, current, baselined, stable, and released.
Software Design	Documentation	Software design completed to date is documented.	Software design document is complete, current, baselined, stable, and released.
Software Design	Interfaces	Logical interfaces are designed and are consistent with required standards.	Software interface designs are complete, current, baselined, stable, and released.
Software Design	Hardware Compatibility	The software design completed to date is compatible with target hardware constraints and assumptions.	The software design has been updated in accordance with the actual target hardware.
Software Design	Dependability	Initial implementation of information assurance, safety, and Reliability, Maintainability, Availability (RMA) requirements in the executable architecture validates feasibility of meeting requirements through prototyping, modeling, or simulation.	The implemented software design meets all information assurance, safety, and RMA requirements. The risks associated with any late or unincorporated changes have been assessed, documented and communicated; an acceptable risk handling plan is in place.
Software Design	Fault Management	The initial implementation of fault and error management in the executable architecture is defined and validated for feasibility through prototyping, modeling, or simulation.	The as-built fault and error management design and code is robust, complete, current, baselined, stable, and released.
Software Design	Flight Software Upload	The initial design of flight software upload capability (e.g., ground and on orbit) in the executable architecture is demonstrated to meet system needs through prototyping, modeling, or simulation.	The as-built flight software upload capability is complete, current, baselined, stable and released and has been demonstrated to meet system needs.
Software Design	Performance	The initial implementation of the executable architecture validates feasibility of meeting performance requirements through prototyping, modeling, or simulation.	The as-built software design meets all performance requirements. The risks associated with any late or unincorporated changes have been assessed, documented and communicated; an acceptable risk handling plan is in place.
Software Design	Traceability	Bi-directional traceability between software design components and software requirements is complete, baselines, stable, and controlled.	Bi-directional traceability between software design components and software requirements is complete, current, baselines, stable, and controlled.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Source Code	Coding Standards and Practices	A plan exists for demonstrating conformance to the coding standards and practices. Preferably, the conformance to coding standards and practices is enforced by tools. For any code completed by SAR, analysis has been performed that demonstrates that the source code for this build conforms to the program's coding standards and practices.	Analysis has been performed that demonstrates that the source code for this build conforms to the program's coding standards and practices.
Source Code	Reuse Code	Reuse code selected by the SAR has been analyzed to determine a) if it meets the coding standards and practices; b) it fits into the software architecture; c) it can be upgraded to satisfy new requirements, and d) the risk of its use. The documentation of the code has been reviewed to determine its pedigree as part of the assessment of risk.	Reuse code for this build meets the coding standards and practices, fits into the software architecture, and if needed, has been upgraded to satisfy new requirements. Any risk of its use has been mitigated.
Source Code	Consistency	Any code completed by the SAR is consistent with requirements, architecture, and design.	Source code for this build is consistent with requirements, architecture, and design.
Source Code	Quality	For any code completed by SAR, code analysis has been performed to identify memory leaks, vulnerabilities, type mismatches, dead code, etc., using automated structure analysis, static analysis, dynamic analysis, and complexity analysis, as appropriate. The identified risk mitigations are planned or complete.	Code analysis has been performed to identify memory leaks, type mismatches, dead code, etc., using automated structure analysis, static analysis, dynamic analysis, and complexity analysis, as appropriate. The identified risks have been mitigated.
Source Code	Information Assurance	For any code completed by SAR, code analysis has been performed to identify vulnerabilities. Selected COTS software has been analyzed to identify vulnerabilities. The identified vulnerabilities have been documented and plans for mitigation have been developed.	Code analysis has been performed to identify vulnerabilities, and COTS software has been analyzed to identify vulnerabilities. The identified vulnerabilities have been documented and mitigated.
Source Code	Traceability	Any code completed by the SAR is traceable to the software design.	Source code components are traceable to software design.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Unit Test Cases	Coverage	For any code developed by SAR: the unit test cases executed all statements and branches in the code. For any exceptions to this coverage (both new and reused code) the risk has been assessed, documented, and communicated.	The unit test cases executed all statements and branches in the code. For any exceptions to this coverage (both new and reused code) the risk has been assessed, documented, and communicated.
Software Unit Test Documentation	Completeness	For any software unit test cases that have been executed by SAR, the software unit test cases, results, analyses, and other artifacts are complete, current, and controlled.	Software unit test cases, results, analyses, and other artifacts are complete, current, and controlled.
Configuration Item List	Design Validation	For any code developed by SAR, unit test results demonstrate the units have correctly implemented their design.	Unit test results demonstrate the units correctly implemented their design.
Software Unit Integration Test Documentation	Completeness	For software unit integration test cases that have been executed by SAR, software unit integration plans, cases, procedures, reports and artifacts are complete, baselines, stable, and controlled.	Software unit integration plans, cases, procedures, reports and artifacts are complete, current, baselines, stable, and controlled.
Software Unit Integration Test Cases	Interfaces	For any code developed by SAR, interfaces are executed for values within, on and out of bounds and are demonstrated to meet software design and system dependability requirements. Behavior of interfaces between components is accurately reflected in the interface documentation.	Interfaces are executed for values within, on and out of bounds and are demonstrated to meet software design and system dependability requirements. Behavior of interfaces between components is accurately reflected in the interface documentation.
Software Unit Integration Test Results	Software Threads	Software threads within the executable architecture are successfully executed, prototyped, modeled or simulated end-to-end for both nominal and off-nominal conditions.	Software threads allocated to the build have been successfully executed end-to-end for both nominal and off-nominal conditions.
Software/ Hardware Integration Test Plan	Hardware Compatibility	General software/hardware integration concepts are defined and documented.	The executable object code is compatible with the target hardware. The software/hardware integration and test plan is complete, current, baselines, stable, and controlled.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software/ Hardware Integration Test Cases	Interfaces	For any code developed by SAR, the test cases include execution of interfaces for values within, on and out of bounds and demonstration that the interfaces meet software design and system dependability requirements. The test cases include demonstration that the behavior of the interfaces, including timing, between hardware and software items satisfies the interface documentation.	Interfaces are executed for values within, on and out of bounds and are demonstrated to meet software design and system dependability requirements. The behavior of the interfaces, including timing, between hardware and software items satisfies the interface documentation.
Software/ Hardware Integration Test Cases	Test Like You Fly	Plans for software TLYF and DITL testing in the software/hardware integration test environment are defined. For critical software, analyses and testing in a representative hardware and software environment has validated the ability of the architecture to satisfy the software TLYF and DITL cases.	Software TLYF and DITL scenarios have been executed successfully in the software/hardware integration test environment.
Software/ Hardware Integration Test Cases	Stress Testing	Plans for the software stress testing in the software/hardware integration test environment are defined. For critical software, analyses and testing in a representative hardware and software environment has validated the ability of the architecture to satisfy the software stress cases.	Software stress scenarios have been successfully executed for CPU and data-bus throughput, concurrency, simultaneous services, and long duration stability testing in the software/hardware integration test environment. Software stress scenarios have been successfully executed within, at and above the expected (designed) performance requirements.
Software/ Hardware Integration Test Cases	Dependability	Plans for the software dependability testing in the software/hardware integration test environment are defined. For critical software, analyses and testing in a representative hardware and software environment has validated the ability of the architecture to satisfy the software dependability cases.	Software fault management, fault tolerance and graceful degradation scenarios have been successfully executed in the software/hardware integration test environment.
Software/ Hardware Integration Test Documentation	Completeness	Top-level software/hardware integration plans are defined and documented.	Software/hardware integration plans, cases, procedures, reports and artifacts are complete, current, stable, baselined and controlled.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software/ Hardware Integration Test Results	Software Threads	Software/hardware threads have been defined and successfully prototyped, modeled or simulated end-to-end for both nominal and off-nominal conditions.	Software/hardware threads allocated to the build have been successfully executed end-to-end for both nominal and off-nominal conditions.
Software Test Plan	Regression Testing	The software test plan clearly describes how regression testing will be performed for any previously verified software that is modified or interfaces with new or modified code. The software test plan describes the plans for developing a suite of regression tests for this software that exercises all functions in both nominal and off-nominal test cases.	The software test plan contains regression tests for any previously verified software that is modified or interfaces with new or modified code. The software test plan describes a suite of regression tests for this software exercising all functions in both nominal and off-nominal test cases. This suite of regression tests has been developed and tested.
Software Test Plan	Verification Plans	Verification methods are adequate to verify the software requirements. The plan for verification is realistic and will fully verify each software requirement. Verification methods in the software test plan match the verification methods documented in the software requirements and interface requirements specification.	The verification methods in the software test plan are complete, current, baselined, stable, and released. The plan for verification is realistic and fully verifies each software requirement.
Software Test Plan	Software Under Test	The software test plan explicitly defines the software under test. If the software is verified incrementally, the plans for regression testing of requirements verified during incremental testing are clearly defined.	The defined software under test is consistent with the as-built software. The software test plan continues to contain plans for regression testing of requirements verified during incremental development.
Software Test Plan	Fidelity of Verification Environments	Software verification environments defined in the test plan have sufficient fidelity to verify the software requirements and are representative of the operational environment (e.g., all interfaces for the software with operational data rates, protocols, and timing).	The as-built software verification environments are consistent with the software test plan.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Test Plan	Maintenance of Verification Environments	Plans exist for hardware preventive maintenance and calibration of the software verification environments. Any hardware in use at SAR has received maintenance and/or calibration updates as needed.	Any required preventative maintenance and/or calibration has been performed for all software test environments.
Software Test Plan	Anomaly Handling	The software test plan clearly describes how test anomalies will be handled. Discrepancy reports are required for each anomaly encountered. A test log entry is required to document each test deviation. The software test plan also clearly describes how retesting will be handled.	The software test plan clearly describes how test anomalies and retesting are handled, including discrepancy reporting and test logging.
Software Test Plan	Traceability	Bi-directional traceability between the software requirements and the tests defined in the software test plan is documented.	Bi-directional traceability between the software requirements and the tests defined in the software test plan is documented.
Software Test Plan	Completeness	The software test plan is defined, documented and controlled depending on the development life cycle model.	All updates to the software test plan are completed, current, baselined, stable, and released. The updates have not compromised the full verification of the software requirements.
Software Test Procedures	Objectives	For any code developed by SAR, software test objectives have been clearly and comprehensively stated to achieve the goals of the test plan for the test to which the code belongs.	Objectives of each qualification test case are clearly and comprehensively stated and achieve the goals of the test plan for the test to which each test case belongs.
Software Test Procedures	Test Cases	For any code developed by SAR, test cases exist that cover both nominal and off-nominal scenarios and, stress input and output data rates and concurrency scenarios. For critical functions, test cases addressing scenarios outside the operational boundary are included.	Test cases exist that cover both nominal and off-nominal scenarios and, stress input and output data rates and concurrency scenarios. For critical functions, test cases addressing scenarios outside the operational boundary are included.
Software Test Procedures	Test Like You Fly	Software/hardware threads (e.g., operations concept) for TLYF and DITL scenarios are defined.	Test cases are representative of operations (e.g., TLYF, DITL, day and year roll-overs). The test cases include operational constants, databases, data rates, and an operationally representative hardware configuration.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Test Procedures	Flight Software Upload	Software/hardware threads (e.g., operations concept) for flight software uploads are defined.	Test cases exist that demonstrate the ability to upload changes to the flight software, including the correct operation of the software after upload and the ability to roll back to the previous software version.
Software Test Procedures	Regression Testing	For any re-use code or code developed by SAR, the software test procedures include procedures for performing regression testing and are prepared according to the test plan.	The software test procedures include procedures for performing regression testing and are prepared according to the test plan.
Software Test Procedures	Inputs	For any code developed by SAR, inputs to each test procedure have been defined and are recognized by the test equipment, including data values, accuracy, precision, formats, data rates, file sizes, and timing.	Inputs to each test procedure have been updated in accordance with as-built code and are recognized by the test equipment, including data values, accuracy, precision, formats, data rates, file sizes, and timing.
Software Test Procedures	Outputs	For any code developed by SAR, output data from each test procedure are fully specified, including data values, accuracy, precision, formats, data rates, and timing. Output data is observable, and sufficient output data is recorded for performing the verification. The results are recorded and clearly demonstrate satisfaction or failure.	Output data from each test procedure is fully specified, including data values, accuracy, precision, formats, data rates, and timing. Output data is observable, and sufficient data is recorded for performing the verification. The results are recorded and clearly demonstrate satisfaction or failure.
Software Test Procedures	Repeatability	For any test procedures developed by SAR, the software test procedures for each test case are repeatable. Test setup, execution, and post-test clean-up steps are well defined.	The software test procedures for each test case are repeatable. Test setup, execution, and post-test clean-up steps are well defined.
Software Test Procedures	Test Procedures	For any test procedures developed by SAR, test procedure steps clearly define the action that the tester must take (stimulus), the response that is supposed to happen, and the pass/fail criteria.	Test procedure steps clearly define the action that the tester must take (stimulus), the response that is supposed to happen, and the pass/fail criteria.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Test Procedures	Pass/Fail Criteria	For any test procedures developed by SAR, pass criteria and failure criteria are clearly and unambiguously specified in each test procedure step or sequence of steps. Content and structure of input data are designed so that failures are clearly observable.	Pass and failure criteria are clearly and unambiguously specified in each test procedure step or sequence of steps. Content and structure of input data are designed so that failures are clearly observable.
Software Test Procedures	Post Test Analysis	For any test procedures developed by SAR, post-test processing and data analysis pass/fail verification criteria are documented and repeatable.	Post-test analysis steps and pass/fail verification criteria are documented and repeatable.
Software Test Procedures	Traceability	For any test procedures developed by SAR, bi-directional traceability between the test cases and software requirements to be verified in each test case is documented. For test cases verifying multiple requirements, bi-directional traceability between the software requirements and the test procedure steps where the requirements are verified is documented.	Bi-directional traceability between the test cases and software requirements to be verified in each test case is documented. For test cases verifying multiple requirements, bi-directional traceability between the software requirements and the test procedure steps where the requirements are verified is documented.
Software Test Procedures	Completeness	The software test procedures (prepared to date, if any) are in accordance with the associated tests defined in the test plan. The software test procedures are complete, baselines, stable, and controlled.	The software test procedures for the build are in accordance with the software test plan and are complete, current, baselines, stable, and controlled.
Software Verification Environment	Validation	Plans exist to validate the software verification environment before the start of software qualification test.	The software verification environment was validated prior to the start of software qualification test to ensure it performs as needed.
Software Verification Environment	Maintenance	Plans exist to perform all necessary maintenance and calibration of hardware in the software verification environment before the start of software qualification test.	The hardware in the verification environments has had its routine preventive maintenance and all necessary calibration was performed prior to the start of software qualification testing. Any necessary maintenance or calibration needed during software qualification testing was performed in a controlled manner.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Verification Environment	Databases	Plans exist to validate the contents of the databases used in the software verification environment before the start of software qualification test.	All databases that are used in verifying the software requirements were controlled and validated prior to the start of software qualification test to ensure their contents are sufficient to verify the requirements.
Software Test Results	Procedure Execution	N/A	The software test procedures have been successfully executed, meet their objectives and verify their allocated requirements. The requirements verification document has been updated to reflect the testing results of this build.
Software Test Results	Regression Testing	N/A	Regression testing of software requirements has been performed for any previously verified software that has been modified.
Software Test Results	Regression Testing for Incrementally Verified Requirements	N/A	Regression testing of software requirements verified incrementally was performed with the complete set of software executing in an operationally representative hardware configuration.
Software Test Results	Requirements Verification	N/A	The software requirements have been fully verified according to the plan. For each test case, the verification of the requirements assigned to the test case is clearly supported by test results and post-test analysis.
Software Test Results	Verification by Analysis Method	N/A	For requirements that are verified by analysis, the analysis steps and results are clearly documented. The rationale for asserting that the analysis shows the requirement(s) are verified is documented and well understood.
Software Test Results	As-Run Procedures	N/A	As-run procedures, including redlines, are controlled. This includes procedures executed initially, procedures executed for any retesting, and procedures executed for regression testing.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Test Results	Log and Summary Documentation	N/A	A chronological log of the actual test execution was maintained. A summary of the results of each test case was documented, including the version of the software under test, version of the test environment, test deviations or problems that occurred, and significant anomalies encountered. This documentation was prepared for all software qualification test sessions, including initial testing, any required retesting, and regression testing.
Software Test Results	Test Anomaly Investigation	N/A	For each test case, rationale explaining why test anomalies and their resolutions, including workarounds, have not compromised the integrity of the test case and the verification of its associated requirements is documented.
Software Test Results	Test Anomaly Resolution	N/A	For each test case, all anomalies have been documented on discrepancy reports and have been resolved. Discrepancies with cause unknown (e.g., unable to duplicate) or resolution of “use as is” have been fully investigated and residual risk has been evaluated; the risks have been communicated to the receiving organization.
Software Test Results	Deviations	N/A	All deviations from the test procedure have been documented, and residual risk has been evaluated.
Software Test Results	Deviations and Waivers	N/A	All deviations and waivers for unverified requirements have been approved. The deviations and waivers have been communicated to the receiving organization.
Software Test Results	IV&V	N/A	All actions have been resolved from any independent verification and validation activities.
Software Version Description	Software Product Installation Instructions	N/A	The installation instructions cover all types of installation files: COTS, configuration files, data files, databases, scripts, filters, and executables.

Product	Area	Early Lifecycle Criteria Software Architecture Readiness (SAR)	Late Lifecycle Criteria Build Turnover Readiness (BTR)
Software Version Description	Site-Specified Installation Instructions	N/A	Installation instructions are clearly documented and validated for every receiving environment and site-specific policies and procedures. Installation scripts for each environment and site are complete, current, baselined, stable, and released.
Software Version Description	Build Scripts	N/A	The values of the check sums, file sizes, or other method for verifying installation of the correct version of the software have been validated against the values produced by the as-built build scripts.
Operational Databases	Configuration Management	Plans exist for configuration management of databases containing operational data at multiple sites, user equipment, and for multiple satellites (e.g., data at the factory, for ground development and operational facilities, and on orbit for one or more satellites).	The initial operational database is baselined and released at the factory and ground development sites.
Operational Databases	Procedures	N/A	Procedures for updating the content of the operational databases and synchronizing the content across the satellite factory, user equipment, ground development and operational facilities, and one or more on-orbit satellites are complete, current, baselined, stable, and released.
Operational Databases	Compatibility Demonstration for Space Target Processor	N/A	The ability to update the content of the operational databases at the satellite factory and to upload the new contents to the target hardware has been demonstrated, including the correct operation of the software post-upload.
Operational Databases	Compatibility Demonstration for Ground Target Processor	N/A	The ability to update the content of the ground database at the ground development facility from the factory operation database and to send commands with the new values has been demonstrated.

Table 3. Assessment Criteria for the Process Perspective

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software development planning, monitoring, and controlling	Software development planning	A process exists for the development of software-related program plans that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires the development and use of program plans that define software program activities (e.g., software development plan; software configuration management plan; software quality assurance plan; software measurement plan; software product evaluation plan). The plans are integrated across all teammates and disciplines and comply with contractual standards and organizational standards and practices. Plans are effective, followed, released, maintained, and updated to reflect changes that impact software (e.g., requirements; scope; constraints; budget; schedule). Deviations from the software planning process are identified and corrective actions applied and tracked to closure.	The program's process for software development planning continues to be followed, controlled, maintained, and updated to reflect changes that impact software. Plans continue to be followed, released, maintained, and updated to reflect changes that impact software (e.g., requirements; scope; constraints; budget; schedule). Deviations from the software planning process are identified and corrective actions applied and tracked to closure.
Software development planning, monitoring, and controlling	Software development monitoring and controlling	A process exists for software development monitoring and controlling that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires that software program management status progress against the software-related program plans, then take appropriate corrective actions when the program's software performance deviates significantly from the plans. Progress is statused per the software-related program plans. Deviations from plans are identified and corrective actions applied and tracked to closure.	The program's process for software development monitoring and controlling continues to be followed, controlled, maintained, and updated to reflect changes that impact software. Progress continues to be statused per the software-related program plans. Deviations from plans are identified and corrective actions applied and tracked to closure.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software development planning, monitoring, and controlling	Software development process planning	A process exists for the development of the program's software processes that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires the development and use of program processes, procedures, standards, and practices. Periodic process assessments demonstrate adherence to defined program processes. The processes, procedures, standards, and practices are integrated across all teammates and disciplines. The processes, procedures, standards, and practices provide software personnel with adequate direction to perform their assigned tasks. The processes, procedures, standards, and practices are defined to the level of "how to" perform as well as "what to" perform. The applicable processes, procedures, standards, and practices are used in the development and test of the software. Processes, procedures, standards, and practices are updated and controlled to reflect changes that impact software. Processes, procedures, standards, and practices are improved by fixing defects in the processes, procedures, standards, and practices or by implementing process improvements. Lessons learned from this or other programs are incorporated into process improvement plans.	The program's process for software development process planning continues to be followed, controlled, maintained, and updated to reflect changes that impact software. The applicable processes, procedures, standards, and practices were used in the development and test of the software. Periodic process assessments continue to demonstrate adherence to defined program processes, procedures, standards, and practices. Processes, procedures, standards, and practices are updated and controlled to reflect changes that impact software. Processes, procedures, standards, and practices are improved by fixing defects in the processes, procedures, standards, and practices or by implementing process improvements. Deviations from documented processes, procedures, standards, and practices are identified, approved, corrective actions applied and tracked to closure.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software development planning, monitoring, and controlling	Software stakeholders	A process exists for the development of a software stakeholder plan that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires the development and use of a software stakeholder plan that identifies relevant stakeholders and their involvement in the software program and is consistent with the overall program plan. Software stakeholder activities are integrated with the program plans. Software stakeholders may be internal or external to the program. The software stakeholder plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	The program's process for software stakeholder planning continues to be followed, controlled, maintained, and updated to reflect changes that impact software. The software stakeholder plan continues to be effective, followed, released, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software development planning, monitoring, and controlling	Software builds	A process exists for the development of a software master build plan that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires the development and use of a software master build plan that defines the sequence in which software components are integrated and tested, supports the system integration and test plan, and is consistent with the software requirements and architecture. The build plan identifies the resources required to develop and test each build (e.g., special test equipment). The build plan provides for early infrastructure builds and early non-developmental item (NDI) builds, as applicable. Currently defined software requirements are allocated to builds in the build plan. If a requirement is only partially implemented in a build, the build plan should clearly state the limitations of the build in meeting that requirement. Interfaces to the software, for example, commands, messages, and responses as defined in an ICD, have been allocated to builds. The build plan supports the end-to-end execution of the software threads. The software master build plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	The program's process for software build planning continues to be followed, controlled, maintained, and updated to reflect changes that impact software. The software master build plan continues to be effective followed, released, maintained, and updated to reflect changes that impact software. The as-built software correctly reflects the build plan. Requirements allocation to builds has been tracked and deviations identified, approved and tracked to closure.
Integrated development environment	Software development environment	A process exists for the development of the software development environment that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process results in a software development environment that supports the program's processes and tools and is integrated across all teammates and disciplines. The software development environment is controlled and updated to reflect changes that impact software.	The program's process for the development of the software development environment continues to be followed, controlled, maintained, and updated to reflect changes that impact software. The software development environment continues to be controlled and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Integrated development environment	Software integration and test environment	A process exists for the development of the software integration and test environments (e.g., development workstation, software simulation of hardware, hardware-in-the-loop, and the target system) that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process results in software integration and test environments that support the program's processes and tools and are integrated across all teammates and disciplines. The process handles resolution of competing needs for verification facility support. The software integration and test environments are controlled and updated to reflect changes that impact software.	The program's process for the development of the software integration and test environment continues to be followed, controlled, maintained, and updated to reflect changes that impact software. The software integration and test environments continue to be controlled and updated to reflect changes that impact software.
Integrated development environment	Software development library	A process exists for the development of the software development library that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process results in a software development library that is integrated across all teammates and disciplines. The software development library is controlled and updated to reflect changes that impact software development.	The program's process for the development of the software development library continues to be followed, controlled, maintained, and updated to reflect changes that impact software. The software development library continues to be controlled and updated to reflect changes that impact software.
Integrated development environment	Software development files	A process exists for each teammate for the development of software development files (SDFs) that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. Any common SDF processes are integrated across teammates and disciplines. The SDFs are controlled and updated to reflect changes that impact software.	The program's process for the development of software development files continues to be followed, controlled, maintained, and updated to reflect changes that impact software. The SDFs continue to be controlled and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Integrated development environment	Non-developmental software	A process exists for the development of a refresh plan for software non-developmental items (NDI) that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. NDI include COTS, GOTS, reuse and open source software. The process requires the development and use of a refresh plan that defines each NDI, how it has been configured and populated for the program's software development environment, and identifies other products with which it interfaces. The COTS refresh plan includes the plan for COTS configuration management. The refresh plan is consistent with the program's processes and tools, is integrated across all teammates and disciplines, and has criteria for determining when NDI must be upgraded to a new version. The COTS refresh plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	The program's process for the development of a refresh plan continues to be followed, controlled, maintained, and updated to reflect changes that impact software. Software NDI have been correctly configured and populated for the program's software development environment. NDI have been upgraded as planned. The refresh plan continues to be effective followed, released, maintained, and updated to reflect changes that impact software.
Systems engineering	System requirements development	A process exists for the development of system requirements that is effective, followed, controlled, and maintained. The process requires that software engineers participate in the development and allocation of system requirements to ensure that the allocated requirements can be implemented by the software.	Updates to the system requirements development process maintain the participation of software engineers.
Systems engineering	System operations concept development	A process exists for the development of the system operations concept that is effective, followed, controlled, and maintained. The system operations concept development process requires participation from software engineers to ensure that the operations concept can be effectively implemented by the software.	Updates to the system operations concept development process maintain the participation of software engineers.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Systems engineering	System architecture development	A process exists for the development of the system architecture that is effective, followed, controlled, and maintained. The process includes a formal decision analysis and resolution process that evaluates identified alternatives against established criteria. The process requires that software personnel are involved in the decisions of what functionality is implemented in hardware versus software, what functionality is allocated to space versus ground, the selection of the on-board and ground processors and other software-sensitive hardware elements, such as redundancy, buffering and interfaces, and the allocation of functionality across space subsystems and ground elements.	Updates to the system architecture development process maintain the participation of software engineers.
Systems engineering	Information assurance	A process exists for the development and use of an information assurance plan that is effective, followed, controlled, and maintained. The process requires that software security experts participate in the development and allocation of information assurance requirements to ensure that the allocated requirements can be implemented by the software. The process requires that all stakeholders, especially any external accrediting agency, have been involved appropriately in software architecture development and review. The information assurance plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	Adherence to the information assurance plan is reflected in the as-built software. The information assurance plan continues to be effective, followed, released maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Systems engineering	Dependability	A process exists for the development and use of a system dependability plan that is effective, followed, controlled, and maintained. The process requires that software dependability experts participate in the development and allocation of dependability requirements to ensure that the allocated requirements can be implemented by the software. The plan requires that data be collected during integration and qualification testing to support the dependability analyses. The dependability plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	Adherence to the dependability plan is reflected in the as-built software. The dependability plan continues to be effective, followed, released, maintained, and updated to reflect changes that impact software.
Systems engineering	Human systems integration	A process exists for the development and use of a human systems integration (HSI) plan that is effective, followed, controlled, and maintained. The process requires that software HSI experts participate in the development and allocation of HSI requirements to ensure that the allocated requirements can be implemented by the software. The HSI plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	Adherence to the HSI plan is reflected in the as-built software. The HSI plan continues to be effective, followed, released, maintained, and updated to reflect changes that impact software.
Systems engineering	Criticality analysis	A process exists for system criticality analysis that is effective, followed, controlled, and maintained. The process requires the development and use of a system criticality analysis report that identifies critical system requirements. The process requires that software criticality analysis experts participate in the development of the system criticality analysis report. The system criticality analysis report is effective, followed, released, maintained, and updated to reflect changes that impact software. Critical system requirements have been flowed to software.	The system criticality analysis is reflected in the as-built software. The criticality analysis process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. Updates to the program's system criticality analysis process maintain the participation of software criticality analysis experts. The system criticality analysis report continues to be effective, followed, released, maintained, and updated to reflect changes that impact software. Critical requirements have been especially well tested.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Systems engineering	System safety	A process exists for system safety analysis that is effective, followed, controlled, and maintained. The process requires the development and use of a system safety plan. The process requires that software safety experts participate in the development and allocation of system safety requirements to ensure that the allocated requirements can be implemented by the software and participate in system safety analyses to ensure that the analyses appropriately include software. The system safety plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	The system safety analysis process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. Adherence to the system safety plan is reflected in the as-built software. The system safety plan continues to be effective, followed, released, maintained, and updated to reflect changes that impact software.
Software engineering	Non-developmental software	A process exists for the identification, evaluation, and selection of NDI that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes defining criteria for the identification, evaluation, and selection of NDI and analysis to determine if the selected NDI meets all evaluation criteria and allocated requirements and provides a low risk solution. NDI needed for early builds are identified and selection rationale provided. The process requires the development and use of a refresh plan that defines each NDI, how it has been configured and populated for the operational environment, and identifies other products with which it interfaces. The refresh plan is consistent with the program's processes and tools, is integrated across all teammates and disciplines, and has criteria for determining when NDI must be upgraded to a new version. The refresh plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	The NDI process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. NDI have been adequately tested and demonstrated to meet allocated requirements and program needs. NDI have been upgraded as planned. The refresh plan continues to be effective, followed, released, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software engineering	Specialty engineering	A process exists for specialty engineering analysis that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes software criticality analysis, dependability, HSI, information assurance and safety. The process includes deriving appropriate software specialty engineering requirements, performing performance analyses to ensure that software specialty engineering requirements can be met by the software architecture and design, and performing specialty engineering testing. The analyses have been updated to reflect changes that impact software.	The software specialty engineering analysis process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. The analyses have been updated to reflect changes that impact software. Software specialty engineering analyses and testing have been performed to demonstrate that software specialty engineering requirements have been met for this build. The analyses have been updated to reflect changes that impact software.
Software engineering	Specialty engineering standards and practices	A process exists to define the program's software specialty engineering standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The specialty engineering standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. Program personnel follow the standards and practices while performing the software specialty engineering activities. The specialty engineering standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software.	The specialty engineering standards and practices continue to be effective, followed, baselined, maintained, and updated to reflect changes that impact software. Adherence to the specialty engineering standards and practices is reflected in the as-built software. The specialty engineering analyses have been updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software requirements	Software requirements development	A process exists for software requirements development that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires a formal decision analysis and resolution process that evaluates identified alternatives against established criteria. Requirements allocation decisions are documented. The process ensures that software requirements have been elaborated to address software functional, performance, non-functional, HSI, behavioral, data integrity, information assurance, safety, dependability, and maintenance requirements, and software standards and practices. The process provides for requirements to be updated to reflect changes that impact software. The process requires a product evaluation to be performed on software requirements and any significant changes to the software requirements.	The program's software requirements development process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.
Software requirements	Software requirements development standards and practices	A process exists for the development of the program's software requirements development standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The software requirements development standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software. The requirements development standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. The standards and practices include elicitation and validation of requirements from requirements stakeholders. Program personnel follow the standards and practices while performing the software design activities.	Adherence to the software requirements development standards and practices is reflected in the as-built software. The program's software requirements development standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software requirements	Software requirements management	A process exists for software requirements management that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires bi-directional traceability between software requirements and parent requirements, and between software requirements and the software architecture. The process requires identification of software test methods for each requirement. The process requires software requirements allocation to builds. The process provides for requirements to be updated to reflect changes that impact software, including their traceability and verification method. The process requires software requirements traceability to be independently reviewed.	Adherence to the software requirements management process is reflected in the as-built software. The program's software requirements management process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.
Software requirements	Software requirements management standards and practices	A process exists to define the program's software requirements management standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The software requirements management standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software. The requirements management standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. The standards and practices include developing and maintaining bi-directional traceability between software requirements and their parent and between software requirements and the software architecture. Program personnel follow the standards and practices while performing the software architecture activities.	Adherence to the software requirements management standards and practices is reflected in the as-built software. The program's software requirements management standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software architecture	Software architectural design	A process exists for the development of the software architecture process that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes a formal decision analysis and resolution process that evaluates identified alternatives against established criteria. The process requires the software architecture to support the system design, system operations concept, and quality attributes, and the architecture decisions to be documented. The process provides for the software architecture to be updated to reflect changes that impact software. The process requires a product evaluation to be performed on the software architecture and any significant changes to the software architecture.	Adherence to the software architecture is reflected in the as-built software. The software architecture has been updated to reflect changes that impact software. The software architecture process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.
Software architecture	Software architecture standards and practices	A process exists to define the program's software architecture standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The software architecture standards and practices are effective, followed, baselined, maintained, and updated to reflect changes that impact software. The architecture standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. The standards and practices include establishment of architecture principles.	Adherence to the software architecture standards and practices is reflected in the as-built software. The program's software architecture standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software design	Software detailed design	A process exists for software detailed design that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes a formal decision analysis and resolution process that evaluates identified alternatives against established criteria. The process requires that the software design is consistent with the software architecture and that design decisions are documented. The process provides for software design to be updated to reflect changes that impact software. The process requires a product evaluation to be performed on the software design.	Adherence to the software detailed design is reflected in the as-built software. The software design has been updated to reflect changes that impact software. The software design process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.
Software design	Software detailed design standards and practices	A process exists for the development of the program's software design standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The software design standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software. The design standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. The standards and practices include establishment of design patterns for common components and services.	Adherence to the software detailed design standards and practices is reflected in the as-built software. The software detailed design standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.
Software implementation	Software coding	A process exists for software coding that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process is consistent with all required specifications, standards, and constraints. The process requires a product evaluation to be performed on the code. The process addresses updating code due to changes that impact software.	The code has been updated to reflect changes that impact software. The software coding process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software implementation	Software coding standards and practices	A process exists for the development of the program's coding standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software. The standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. The coding standards and practices are compatible with the software specialty engineering standards and practices.	Adherence to the coding standards and practices is reflected in the as-built software. The coding standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.
Software implementation	Software code analysis	A process exists for code analysis that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The processes includes automated structure analysis, static analysis, dynamic analysis, complexity analysis, and analysis for memory leaks, vulnerabilities, type mismatches, and dead code. The process for code analysis enforces the coding standards and practices. The process includes criteria for determining when a specific analysis is appropriate.	Code analyses have been performed to demonstrate that software requirements have been met for this build. The analyses have been updated to reflect changes that impact software. The program's code analysis process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.
Software implementation	Software unit testing	A process exists for unit testing that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes the preparation of test cases, test procedures, test conduct, and test reports. The process addresses initial test, re-test, and regression test. The process requires a product evaluation to be performed on the software unit test cases. The process addresses updating unit test cases and procedures due to changes that impact software.	Test cases and procedures have been updated to reflect changes that impact software. The program's software unit test process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software implementation	Software unit testing standards and practices	A process exists for the development of the program's software unit test standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software. The standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. The standards and practices address unit test coverage requirements. The standards and practices establish unit test requirements for nominal and off-nominal test cases. The standards and practices are compatible with the software specialty engineering standards and practices.	The program's software unit testing standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.
Software unit integration and testing	Software executables	A process exists for building the executable software that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes all activities necessary to prepare executables from configuration managed code for all integration test platforms, including the target platform. The process addresses updating software executables due to changes that impact software.	The program's process for building executable software continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.
Software unit integration and testing	Software unit integration testing	A process exists for software unit integration testing that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes the preparation of test plans, test cases, test procedures, test conduct, and test reports. The process addresses initial test, retest, and regression test. The process addresses updating integration test cases and procedures due to changes that impact software.	The program's process for software unit integration testing continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software unit integration and testing	Software unit integration testing standards and practices	A process exists for development of the program's software unit integration testing standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software. The standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. The standards and practices address software internal interface coverage and boundary conditions. The standards and practices establish unit integration testing requirements for nominal and off-nominal test cases. The standards and practices are compatible with the software specialty engineering standards and practices.	The program's software unit integration testing standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software qualification testing	Software qualification testing	A process exists for software qualification testing that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes the preparation of test plans, test cases, test procedures, test conduct, and test reports. The process addresses initial test, retest and regression test. The process includes incorporating test like you fly, day-in-the-life, duration, and other operational tests, as appropriate. The process requires that qualification test is performed by personnel independent of the software developers. The process addresses the involvement of external stakeholders, such as customers, quality assurance, and readiness assessors. The process requires updating qualification test cases and procedures due to changes that impact software and re-running the qualification tests. These changes include software changes, environment changes, and operational concept changes.	The program’s process for software qualification testing continues to be effective, followed, released, maintained, and updated to reflect changes that impact software.
Software qualification testing	Software qualification testing standards and practices	A process exists for development of the program’s software qualification testing standards and practices that is effective, followed, maintained, and updated to reflect changes that impact software. The standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software. The standards and practices include government, commercial, and international standards as appropriate for the program’s contract, scope, and constraints. The standards and practices require a product evaluation to be performed on software qualification test plans. The standards and practices require a test readiness review prior to software qualification test. The standards and practices require software quality assurance participation in software qualification testing.	The program’s software qualification testing standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software qualification testing	Software uploads	A process exists to create, test, qualify, and upload flight software on the operational vehicle that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires that the software be re-qualified before upload. The process addresses roll-back requirements, if applicable. The upload and rollback products and procedures, if any, must be qualified.	The program's process for software uploads continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.
Software/ hardware integration and testing	Software/ hardware integration and testing	A process exists for software/hardware integration and testing that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes the preparation of test plans, test cases, test procedures, test conduct, and test reports. The process address initial test, retest and regression test. The process includes incorporating test like you fly, day-in-the-life, duration, and other operational tests, as appropriate. The process includes performance tuning. The process addresses updating test cases and procedures due to changes that impact software.	The process for software/hardware integration and testing continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software/ hardware integration and testing	Software/ hardware integration and testing standards and practices	A process exists for development of the program's software/hardware integration and testing standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software. The standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. The standards and practices require a product evaluation to be performed on software/hardware integration and test plans. The standards and practices require a test readiness review prior to software/hardware integration and test. The standards and practices require software quality assurance participation in software/hardware integration and testing.	The program's software/hardware integration and testing standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software transition to operations	Planning software transition to operations	A process exists for planning the software transition to operations that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires the development of a software transition to operations plan that is consistent with the overall program plan. The initial concept for transition to operations is defined (e.g., software will be operated by the government, by the development contractor, or by a third party). The plan includes an evaluation of data rights to ensure that the operational site has access to all required code and documentation. The plan identifies all software products necessary for operations, including the preparation of the executable software, version descriptions for user sites, user manuals, and computer operations manuals. The plan for transition identifies the operational sites and addresses operator training, deployment, and activation strategy, including staged operations and roll-back. The software transition to operations plan is updated and released to reflect changes that impact software.	The program’s software transition to operations plan continues to be effective, followed, released, maintained, and updated to reflect changes that impact software. If this build will transition to operations, the plan is complete and transitioning to operations according to the plan is low risk.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software transition to maintenance	Planning software transition to maintenance	A process exists for planning the software transition to maintenance that that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires the development of a software transition to maintenance plan that is consistent with the overall program plan. The initial concept for transition to maintenance is defined (e.g., software will be maintained by the government, by the development contractor, or by a third party). The plan includes an evaluation of data rights to ensure that the maintenance site has access to all required code and documentation. The plan identifies all software products necessary for maintenance, including executable software, source files, version descriptions for the maintenance site, the "as-built" software design and related materials, updated system/subsystem design description, updated software requirements, updated system requirements, maintenance manuals, computer programming manuals, firmware support manuals. The plan addresses the acquisition or transition of the integrated development environment and the software integration and test environments to the maintenance environment, and training in their use. The plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	The program's software transition to maintenance plan continues to be effective, followed, released, maintained, and updated to reflect changes that impact software. If this build will transition to maintenance, the plan is complete and transitioning to maintenance according to the plan is low risk.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software configuration management	Software configuration management process	A process exists for each teammate for software configuration management planning that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires the development of a software configuration management plan. The process requires configuration identification, configuration control, configuration status accounting, and configuration audit. The software configuration management plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	The program's software configuration management process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. The program's software configuration management plan continues to be effective, followed, released, maintained, and updated to reflect changes that impact software.
Software configuration management	Software configuration management standards and practices	A process exists for development of the program's configuration management standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software. The standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. The standards and practices address software configuration identification, software configuration control, software status accounting, and software configuration audits. The standards and practices require that a product evaluation be performed on the software configuration management plan. The standards and practices require software quality assurance participation in software configuration management.	The program's software configuration management standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software configuration management	Software configuration identification	A process exists for each teammate for the identification of software configuration items that is effective, followed, controlled, maintained, and updated for changes that impact software. The process requires the development of a configuration item list with unique identifiers. The list includes developed items, non-developmental items, and items delivered from suppliers integrated within other items. The process supports multiple concurrent baselines. The process ensures that all products used in test, launch, maintenance, and operations are identified. The configuration item list is updated for changes that impact software.	The program's software configuration identification process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. The software configuration item list is updated to reflect changes that impact software. The as-built software has been baselined.
Software configuration management	Software configuration control	A process exists for each teammate for configuration control of software that is effective, followed, controlled, and maintained, and updated for changes that impact software. The process provides build procedures, baseline control, access control, and data integrity. The process applies to requirements, design, code, documentation, COTS products, and databases. The process provides for roll-back to a known configuration. The process provides for patch management. The process describes how to build executables from code.	The program's software configuration control process continues to be effective, followed, controlled, maintained, and updated for changes that impact software. All required items associated with the as-built software, including test environments, test drivers and scripts, and test data, are under configuration control.
Software configuration management	Software configuration status accounting	A process exists for each teammate for software configuration status accounting that is effective, followed, controlled, and maintained, and updated to reflect changes that impact software. The process provides for efficient and timely performance of status accounting.	The program's software configuration status accounting process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. Configuration status accounting has been regularly performed and is current in accordance with the plan.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software configuration management	Software configuration audits	A process exists for each teammate for software configuration audits that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process provides for efficient and timely performance of configuration audits.	The program's software configuration audit process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. A software configuration audit has been performed prior to BTR per the configuration management plan.
Software assessments	Software product evaluations	A process exists for each teammate for software product evaluations that is effective, followed, controlled, and maintained, and updated to reflect changes that impact software. The process may define different types of software product evaluations with different levels of formality (e.g., peer reviews, inspections, walkthroughs). The process specifies that reviews performed are appropriate for the product and the criticality of the product. The process requires that every software product is evaluated by knowledgeable reviewers and relevant stakeholders, other than the author of the product, against established criteria. For more formal reviews, the process requires that a) action items from software product evaluations are documented and tracked to closure, and b) defects from software product evaluations are documented, categorized by type and severity, and tracked to closure. For less formal reviews, minutes capture the product reviewed, attendees and major decisions made.	The program's software product evaluation process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. The evaluations performed are appropriate for the product and the criticality of the product.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software quality assurance	Software quality assurance process	A process exists for each teammate for software quality assurance that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process requires the development of a quality assurance plan. The process provides for quality assurance participation in software activities. The process provides for efficient and timely performance of quality assurance audits of software products and processes against organizational and contractual standards and procedures. The process requires that the software quality organization have a reporting chain independent from program management. The process defines how quality assurance non-compliance issues are handled. The software quality assurance plan is effective, followed, released, maintained, and updated to reflect changes that impact software.	The program's software quality assurance process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. The program's software quality assurance plan continues to be effective, followed, released, maintained, and updated to reflect changes that impact software. Software quality assurance audits have been performed as planned.
Software quality assurance	Software quality assurance standards and practices	A process exists for development of the program's software quality assurance standards and practices that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The standards and practices are effective, followed, released, maintained, and updated to reflect changes that impact software. The standards and practices include government, commercial, and international standards as appropriate for the program's contract, scope, and constraints. The standards and practices require that a product evaluation be performed on the software quality assurance plan. The standards and practices define the participation of software quality assurance in the software development activities.	The program's software quality assurance standards and practices continue to be effective, followed, released, maintained, and updated to reflect changes that impact software.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Corrective action	Problem resolution and change management	A process exists for each teammate for corrective action that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes both change requests and problem reports. The process describes the corrective action system, and defines the causal analysis (e.g., root cause analysis), and resolution process, including any change control and or corrective action boards. The process requires that all problems and changes are documented and corrective actions are implemented and tracked to closure. The risks have been assessed, documented, communicated and an acceptable risk handling strategy is in place.	The program's software problem resolution and change management process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. Actions from previous reviews are reviewed and closed as appropriate. The risks have been assessed, documented, communicated and an acceptable risk handling strategy is in place. If significant numbers of discrepancies were discovered late in the development, causal analysis was done to identify and remove additional problems which testing may not yet have uncovered.
Process improvement	Software process assessment	A process exists for performing periodic software process assessments that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process applies to all teammates and disciplines in accordance with contract requirements and organizational policies. The process addresses process deviations and changes due to organizational process updates. Previous process assessments have demonstrated adherence to defined program process. Gaps and shortcomings have been addressed and corrective actions are being worked. Process deviations, improvements, and changes since last technical review are documented. The risks have been assessed, documented, communicated and an acceptable risk handling strategy is in place.	The program's software process assessment process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. Previous process assessments have demonstrated adherence to defined program process. Gaps and shortcomings have been addressed and corrective actions are being worked. Process deviations, improvements, and changes since last technical review are presented. The risks have been assessed, documented, communicated and an acceptable risk handling strategy is in place.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Process improvement	Software process monitoring and control	A process exists for performing periodic analyses of process execution metrics that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process includes technical, cost, schedule and quality metrics to identify changes to process that would improve execution performance.	The program's software process monitoring and control process continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software.
Joint technical and management reviews	Joint technical reviews	A process exists for performing joint technical reviews that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process is consistent with contract requirements and organizational policies. The process requires that action items from previous reviews are reviewed prior to SAR, each action item has a closure plan, and that the closure plan is on track. The process requires that relevant stakeholders participate in the activities associated with this review. The process requires that mandatory reviewers be identified and participate in the review. The process requires that a review cannot be closed without the mandatory reviewer input. The process requires that minutes and action items be captured. The process requires that risks are assessed, documented, communicated and an acceptable risk handling strategy is in place.	The program's process for joint technical reviews continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. Action items from previous reviews are reviewed prior to BTR, each action item has a closure plan, and the closure plan is on track. The risks have been assessed, documented, communicated, and an acceptable risk handling strategy is in place.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Joint technical and management reviews	Joint management reviews	A process exists for performing joint management reviews that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process is consistent with contract requirements and organizational policies. The process requires that action Items from previous reviews are reviewed prior to SAR, each action item has a closure plan, and that the closure plan is on track. The process requires that relevant stakeholders participate in the joint management reviews. The process requires that minutes and action items be captured. The process requires that risks are assessed, documented, communicated and an acceptable risk handling strategy is in place.	The program's process for joint management reviews continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. Action items from previous reviews are reviewed prior to BTR, each action item has a closure plan, and the closure plan is on track. The risks have been assessed, documented, communicated and an acceptable risk handling strategy is in place.
Risk management	Software risk management	A process exists for performing software risk management that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process is consistent with contract requirements and organizational policies. The process includes risk identification, risk assessment, risk handling, and risk monitoring and reporting. The process produces a list of software risks. A complete software risk list is identified and appropriate mitigation actions are being taken.	The program's process for software risk management continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. The risk list and risk mitigation actions have been updated to reflect the as-built software. The risks have been assessed, documented, communicated and an acceptable risk handling strategy is in place.
Management indicators	Software metrics	A process exists for planning, collecting, analyzing, and reporting software metrics that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process is consistent with contract requirements and organizational policies. The process provides that deviations from plans are identified and corrective actions are applied and tracked to closure. Metrics reports have been prepared and analyzed and corrective actions have been initiated.	The program's process for software metrics continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. Metrics reports have been prepared and analyzed and corrective actions have been initiated.

Process	Area	Early Life Cycle Criteria Software Architecture Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Supplier management	Software supplier management	A process exists for identifying, evaluating, and selecting qualified suppliers of products and services that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process is consistent with contract requirements and organizational policies, and flows all appropriate contract requirements to the suppliers. The process covers both developers of new products or services and vendors of COTS products or services. The process requires the use of defined criteria in the evaluation and selection process. The process addresses the maintenance of developed and COTS products. The process addresses the acquisition of the appropriate data rights. The process addresses the long-term viability of suppliers and mitigation plans if the supplier is no longer able to supply the product or service. Plans and criteria for supplier selections not yet made are provided.	The program's process for software supplier management continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. Plans and criteria for supplier selections not yet made are provided. The criteria defined and evaluation activities performed to date are appropriate for the product and the criticality of the product.
Training	Training software personnel	A process exists for identifying the training needs of software personnel that is effective, followed, controlled, maintained, and updated to reflect changes that impact software. The process ensures that all personnel get the required training or on-the-job instruction. The process requires that the program maintain training records. The process addresses both organizational and program training, including process, methods, and tools. The training process addresses both the maintenance and increase of the proficiency of the personnel. Training plans are updated and released to reflect changes that impact software. Training records show that training is accomplished as planned.	The program's process for software training continues to be effective, followed, controlled, maintained, and updated to reflect changes that impact software. Training plans are updated and released to reflect changes that impact software. Training records show that training is accomplished as planned.

Table 4. Assessment Criteria for the Resource Perspective

Resource	Area	Early Life Cycle Criteria Software Requirements Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Cost and Schedule Budgets	Execution	Software program is on budget and schedule at SAR or mitigation plan is in place. Sufficient resources exist to complete per schedule including anticipated ECPs. Sufficient budget and schedule time exists for active stakeholder involvement. Plan forward has sufficient budget and schedule for execution including risk reserve.	Software program is on budget and schedule at BTR or mitigation plan is in place. Sufficient resources exist to support subsequent activities, including further integration, maintenance, and operations as applicable.
Cost and Schedule Budgets	Training	Budget and schedule are in place to execute training plan to achieve and maintain proficiency of assigned personnel.	Budget and schedule are in place to execute remaining training required by the training plan to achieve and maintain proficiency of assigned personnel.
Personnel	Clearances	Sufficient security clearance billets exist for both staff and stakeholders. Program has sufficient pool of staff with appropriate security clearances.	Sufficient security clearance billets exist for both staff and stakeholders. Program has sufficient pool of staff with appropriate security clearances.
Personnel	External Support to Software	Adequate system engineering personnel and other SMEs are funded and available to support entire software life cycle.	Adequate system engineering personnel and other SMEs are funded and available to support remaining software life cycle.
Personnel	Organizational structure	Software organizational structure is defined and enables effective functioning as a team, including across corporate boundaries, if applicable.	Software organizational structure is defined and enables effective functioning as a team, including across corporate boundaries, if applicable. Software organizational structure is sufficient for supporting the remaining work scheduled.
Personnel	Roles and Responsibilities	Organizational roles and responsibilities related to software development have been identified, communicated, assigned, understood, and are being performed.	Organizational roles and responsibilities related to software development have been identified, communicated, communicated, assigned, understood, and are being performed. Roles and responsibilities are sufficient for supporting the remaining work scheduled.
Personnel	Skills	Staff is knowledgeable and proficient in the methodology, development environment, NDI, tools, and processes.	Staff is knowledgeable and proficient in the development environment, NDI, tools and processes.

Resource	Area	Early Life Cycle Criteria Software Requirements Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Personnel	Staffing level	Adequate software staff is available to support concurrent development and test phases, including multiple concurrent builds if applicable. Software development and test personnel are available for timely root cause analysis and debugging of issues discovered during testing.	Adequate software staff continues to be available to support concurrent development and test phases, including multiple concurrent builds if applicable, and post-development software maintenance. Software development and test personnel are available for timely root cause analysis and debugging of issues discovered during testing and operations, if applicable.
Personnel	Staffing Profile	Sufficient budget is in place for adequate staff, addressing all defined roles (e.g., management, SW Architect, design, code, test, integration, configuration management). Staffing profile is realistic.	Sufficient budget in place for adequate staff, addressing all defined roles (e.g., management, SW Architect, design, code, test, integration, configuration management). Staffing profile is realistic.
Personnel	Staffing Stability	Turnover of software personnel is sufficiently low that forward progress is not impeded.	Turnover of software personnel is sufficiently low that forward progress is not impeded.
Personnel	Stakeholders	Program has engaged stakeholders early in development of the software requirements and architecture. Stakeholder participation is consistent with stakeholder plan.	Stakeholder participation continues to be consistent with stakeholder plan.
Integrated Development Environment	Architecture & Design Tools	Software architecture and design tools are identified (e.g., UML, Simulink, MATLAB), and sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tools are effective for program development needs.	Tools are effective for program development needs. Sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tool obsolescence has been addressed.
Integrated Development Environment	Collaboration Tools	Collaboration tools are identified (e.g., Portals, technical data, eROOM), and sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tools are effective for program development needs.	Tools are effective for program development needs. Sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tool obsolescence has been addressed.

Resource	Area	Early Life Cycle Criteria Software Requirements Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Integrated Development Environment	Configuration Management/ Data Management Tools	Change management and change control tools are identified (e.g., change management, defect tracking, version control), and sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tools are effective for program development needs.	Tools are effective for program development needs. Sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tool obsolescence has been addressed.
Integrated Development Environment	Debug Tools	Software debug tools are identified (e.g., debugging and tuning, trace, step, breakpoint, profiling), and sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tools are effective for program development needs.	Tools are effective for program development needs. Sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tool obsolescence has been addressed.
Integrated Development Environment	Development Infrastructure	Sufficient workstations, including spares and consumables, are available to support project plans in all development areas. Networks meet project throughput requirements. Security gateways meet security requirements and have sufficient performance to meet project requirements.	Sufficient workstations, including spares and consumables, are available to support project plans in all development areas. Networks meet project throughput requirements. Security gateways meet security requirements and have sufficient performance to meet project requirements.
Integrated Development Environment	Facility	Physical development facilities meet security requirements and are adequate to support all concurrent activities.	Physical development facilities meet security requirements and are adequate to support all concurrent activities.
Integrated Development Environment	Implementation & Build Tools	Implementation and build tools are identified (e.g., compilers, linkers, build scripts), and sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tools are effective for program development needs.	Tools are effective for program development needs. Sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tool obsolescence has been addressed.
Integrated Development Environment	Code analysis Tools	Code analysis tools are identified (see process spreadsheet), and sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tools are effective for program development needs.	Tools are effective for program development needs. Sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tool obsolescence has been addressed.

Resource	Area	Early Life Cycle Criteria Software Requirements Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Integrated Development Environment	Software Management Tools	Software Management Tools are identified (e.g., scheduling, metrics), and sufficient licenses are purchased including maintenance extensions. Technical support is available. Tools are effective for software management needs.	Tools are effective for program management needs. Sufficient licenses are purchased including maintenance extensions. Technical support is available. Tool obsolescence has been addressed.
Integrated Development Environment	Synchronization	Tools selected are compatible across all necessary stakeholders, teams, and subcontractors. Tools across the program are compatible.	Tools selected are compatible across all necessary stakeholders, teams, and subcontractors. Tools across the program are compatible.
Integrated Development Environment	System Engineering Tools	System Engineering tools have been identified (e.g., requirement management, analysis,…), and sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tools are effective for system engineering needs.	Tools are effective for program system engineering needs. Sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tool obsolescence has been addressed.
Integrated development environment	Software development library	The software development library exists and is maintained. The software development library is inclusive of all teammates and disciplines. Training exists in the usage and maintenance of the software development library. The software development library is updated to reflect changes that impact software development.	The software development library has been maintained as planned. Training is kept current.
Integrated development environment	Software Development Files	Software development file (SDF) processes exist, and are effective, followed, and maintained. The software development file processes are updated to reflect changes that impact software development.	Software development files (SDF) have been properly maintained and demonstrate that the applicable software development processes have been adhered to.
Operational Support Environment	Resources	Plan is in place for acquisition and installation of the necessary resources for operational support, including tools, equipment, and facilities.	Adequate tools, equipment and facilities are available to support anomaly analysis and verification. Tool obsolescence has been addressed.

Resource	Area	Early Life Cycle Criteria Software Requirements Readiness (SAR)	Late Life Cycle Criteria Build Turnover Readiness (BTR)
Software Integration and Verification Environment	Resources	Plan is in place for acquisition and installation of the necessary resources for software integration and verification, including tools, equipment, and facilities. Security requirements have been adequately addressed in the plan. The planned software integration and test environment supports the project's processes and tools.	Sufficient equipment, including spares and consumables, is available to support software integration and verification plans. All necessary test tools, simulators, and emulators are available. Networks meet software integration and verification throughput requirements. Security gateways meet security requirements and have sufficient performance to meet software integration and verification requirements.
Software Integration and Verification Environment	Tools	Tools are identified (e.g., test beds, test harness, analysis model, auto generation of test scripts, simulators, emulators), and sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tools are effective for software integration and verification needs.	Tools are effective for software integration and verification needs. Sufficient licenses have been purchased including maintenance extensions. Technical support is available. Tool obsolescence has been addressed.
Software Maintenance Environment	Resources	Plan is in place for acquisition and installation of the necessary resources for software maintenance, including tools, equipment, and facilities. Security requirements have been adequately addressed in the plan.	Adequate tools, equipment, and facilities are available to support software maintenance as applicable to the software life cycle and contract. Tool obsolescence has been addressed.
System Integration and Verification Environment	Resources	Sufficient resources are planned to support software integration and test needs in the system integration and verification environment.	Software resources are available to support software integration and test needs in the system integration and verification environment.

4. Acronyms

ASIC	Application Specific Integrated Circuit
ATAM	Architecture Trade-off Analysis Method
BTR	Build Turnover Review
CDR	Critical Design Review
CM	Configuration Management
CMMI	Capability Maturity Model Integration
CMU	Carnegie Mellon University
CONOPS	Concept of Operations
COTS	Computer Off the Shelf
CSCI	Computer Software Component Item
DITL	Day in the Life
EMD	Engineering and Manufacturing Development (life cycle phase)
FMEA	Failure Mode, and Effects Analysis
FMECA	Failure Mode, Effects, and Criticality Analysis
FPGA	Field-Programmable Gate Array
FRR	Flight Readiness Review
FSM	Functional Size Management
GOTS	Government Off the Shelf
HIL	Hardware in-the-loop
HMI	Human Machine Interface
HIS	Human Systems Integration
I&T	Integration and Test
IA	Information Assurance
IBR	Integrated Baseline Review
ICD	Interface Control Document
ICR	Initial Checkout Review
IEC	International Electro-technical Commission
IEEE	Institute of Electrical and Electronics Engineers
IRS	Interface Requirements Specification
ISBN	International Standard Book Number
ISO	International Organization for Standardization
IV&V	Independent Verification and Validation
KPMs	Key Performance Measures
KPPs	Key Performance Parameters
MIL	Military
MRR	Mission Readiness Review
NDI	Non-developed item includes COTS, GOTS, reuse software, and open source software. NDI can be singular or plural.
O&S	Operations and Support (life cycle phase)
OPSCON	Operations Concept
ORB	Object Request Broker
PDR	Preliminary Design Review
PMBOK	Project Management Body of Knowledge
PSR	Pre-Ship Review
RMA	Reliability, Maintainability, Availability
SAR	Software Architecture Review
SDF	Software Development File
SDR	System Design Review
SEI	Software Engineering Institute
SFR	System Functional Review

SLCM	Software Life Cycle Model
SRA	Software Readiness Assessment
SRR	System Requirements Review
STD	Standard
SQuaRE	Software Product Quality Requirements and Evaluation
SW	Software
TD	Technology Development (life cycle phase)
TLYF	Test like you fly
TOR	Technical Operating Reference
TPM	Technical Performance Measure
TRR	Test Readiness Review
UML	Unified Modeling Language
VCRM	Verification Cross Reference Matrix
VHSIC	Very-High-Speed Integrated Circuit

Appendix A. Glossary

The terms in Table 5 are intended to provide context for the criteria and assessment process where the reader may be unfamiliar with a word or phrase, or where the term may have multiple conflicting interpretations requiring further distinction. Definitions may have been altered from the sources listed to provide additional contextual clarification. Sources are included only to give the user the option to research the topic further and should not be interpreted as a requirement for compliance.

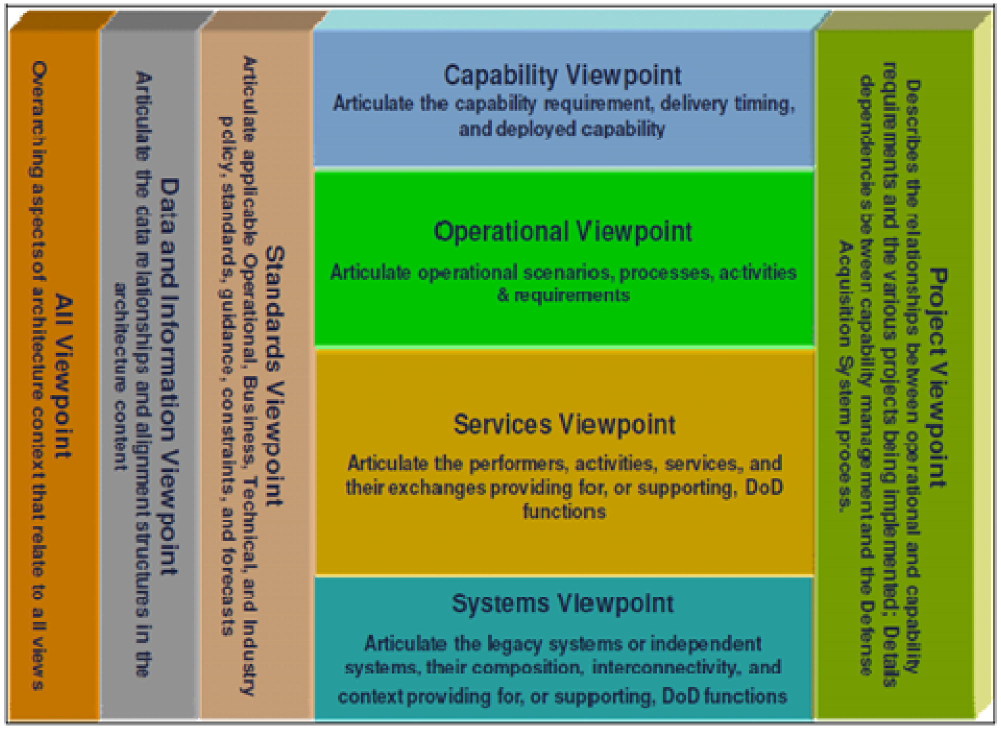
Table 5. Glossary

Term	Definition	Source
Algorithm	(1) A finite set of well-defined rules for the solution of a problem in a finite number of steps. (2) A sequence of operations for performing a specific task. (3) A finite ordered set of well-defined rules for the solution of a problem Example: a complete specification of a sequence of arithmetic operations for evaluating sine x to a given precision.	(1 - 2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (3) (ISO/IEC 2382-1:1993 Information technology--Vocabulary--Part 1: Fundamental terms, 01.05.05) (Example) http://pascal.computer.org/sev_display/index.action
Algorithm Design Document	An algorithm design document provides a description of the algorithms, theoretical basis, mathematical order of computational burden as a function of inputs and stopping criteria, and an outline of the realization in an algorithm design language or pseudo-code appropriate for the software development team.	G. Whittaker
Allocated Requirements	Requirements may be “allocated” to separate builds or increments or to specific architectural elements of an existing or envisioned architecture in a divide and conquer strategy. During requirements analysis and development, typically at the system level a decision is made to “allocate” requirements to software, firmware, or hardware. Various principles and objectives may drive the allocation process such as commonality of operational or functional area, priority of capabilities, risk reduction, incremental effort sizing or to satisfy specific quality attributes such as performance, scalability, reliability, etc.	G. Whittaker
Allocation	(1) The process of distributing requirements, resources, or other entities among the components of a system or program. (2) The result of the distribution of requirements, resources, or other entities among the components of a system or program. (3) The decision to assign a function or decision to hardware, software, or humans. Note: Allocation may be made entirely to hardware, software, or humans, or to some combination, to be resolved upon further functional decomposition.	(1 -2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (3) (IEEE 1220-2005 IEEE Standard for the Application and Management of the Systems Engineering Process, 3.1.3)

Term	Definition	Source
Analysis	Detailed examination of the elements or structure of something, typically as a basis for discussion or interpretation (e.g.,statistical analysis, an analysis of popular culture). For example, an analysis of architectural alternatives compares desired architecture attributes with several candidate architectures in order to identify the architecture that most effectively satisfies the desired attributes through its structure, layers, functional segregation, scalability and expected usage. A statistical analysis of software defects and trends can be made to provide an estimate of latent defects in delivered products that will turn have to be dealt with by subsequent life cycle phases.	Oxford Online Dictionary (http://oxforddictionaries.com/view/entry/m_en_us1221618#m_en_us1221618) Example by G. Whittaker
Analysis Results	Analysis results need to be documented in a form that provides objective evidence for reviewers and assessors. Key elements include statement of purpose and scope of analysis, assumptions and constraints, criteria that were applied for the conduct of the analysis, methodology and experimental design when appropriate, names and roles of people who participated in the analysis, dates for Technical exchange meetings or other key events that supported the development of the analysis.	G. Whittaker
Anomaly	(1) Condition that deviates from expectations, based on requirements specifications, design documents, user documents, or standards, or from someone's perceptions or experiences. (2) Anything observed in the documentation or operation of software or system that deviates from expectations based on previously verified software products, reference documents, or other sources of indicative behavior.	(1) (IEEE 1028-2008 IEEE Standard for Software Reviews and Audits, 3.1) (2) (IEEE 829-2008 IEEE Standard for Software and System Test Documentation, 3.1.4)
Anomaly Handling	Anomaly handling is a term used in multiple domains, including development, test, operations, and fault management. (1) Development and Test activities need to document the anomalous behavior and input conditions that lead to it so that developers can more effectively identify causes and corrective actions. (2) Operators need clear guidelines and procedures for expeditiously addressing anomalies as they occur, so that evidence for root cause analysis is not inadvertently lost while trying to restore normal service after an unexplained anomaly has occurred. (3) Fault management must handle faults in a systematic fashion that ensures graceful degradation of service while protecting mission assets from lethal excursions in state space.	G. Whittaker

Term	Definition	Source
Appraisals	In the CMMI Product Suite, an examination of one or more processes by a trained team of professionals using an appraisal reference model as the basis for determining, at a minimum, strengths and weaknesses. See also "assessment."	CMU/SEI-2006-TR-008, CMMI® for Development, Version 1.2
Architecture	(1) Fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. (2) The organizational structure of a system or component. (3) The organizational structure of a system and its implementation guidelines. Note: sometimes refers to the design of a system's hardware and software components. Syn: architectural structure See also: component, module, subprogram, routine.	(1) (ISO/IEC 15288:2008 Systems and software engineering--System life cycle processes, 4.5) (2 3) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)

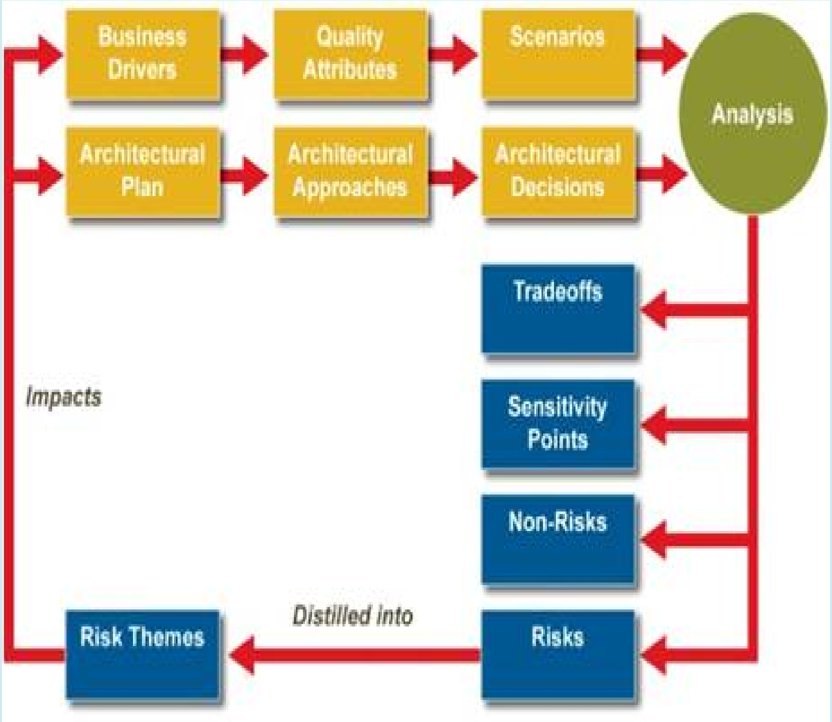
Term	Definition	Source
Architecture Principles	(1) Architecture principles are abstractions of successful strategies, design patterns or solution approaches. Divide and conquer as a general principle has been successfully applied in many domains fraught with overwhelming complexity, and it is the root of many *architectural principles* such as: object oriented design, modular design, layered architecture, n-tiers, and service-oriented architecture. Applying a particular architectural principle establishes a constraint on the design space and consequently has an effect on the quality attributes of the solution. Loose coupling is a principle that promotes maintainable composition of reusable services without undue stress on synchronized governance; conversely, monolithic construction demands tight integration and control of all components to yield optimal performance at the cost of high maintenance effort. (2) Architecture principles can roughly be separated into two sets: functional and constructional principles. This separation is based on the two different notions of systems. Functional principles restrict and guide the functional (behavioral) part of a system; they say something about the black box view of a system. Constructional principles restrict and guide the constructional part of a system; they say something about the white box view of a system. An example of a functional principle can be: The system should expose its functionality through web services. This principle influences the functionality of the system (of all systems built using this architectural principle). This principle also identifies (a part of) the project which should create or change the actual system. An example of a constructional principle can be: The system should consist of modular, reusable parts. This principle influences the quality attributes of a system. It also identifies the project which should create or change the actual system. Constructional principles address topics as availability, usability, security, stability, maintainability, etc. Note: also see ATAM - Architectural Tradeoff Analysis Method	(1) G. Whittaker (2) The Value of Architecture, Johan den Haan (http://www.theenterprisearchitect.eu/archive/2007/10/27/the-value-of-architecture)

Term	Definition	Source
Architecture views or Viewpoints	An architecture view or viewpoint is a selected set of architectural data that has been organized to facilitate visualization in an understandable way. An Architectural Description can be visualized in a number of formats, such as dashboard, fusion, textual, composite, or graphics, which present data and derived information collected in the course of the development of an Architectural Description. A view is only a presentation of a portion of the architectural data, in the sense that a photograph provides only one view of the object within the picture, not the entire representation of that object. Figure 3.4.2-1 provides a graphical representation of the architecture viewpoints in DoDAF V2.0.  All Viewpoint Overarching aspects of architecture context that relate to all views Data and Information Viewpoint Articulate the data relationships and alignment structures in the architecture content Standards Viewpoint Articulate applicable Operational, Business, Technical, and Industry policy, standards, guidance, constraints, and forecasts Capability Viewpoint Articulate the capability requirement, delivery timing, and deployed capability Operational Viewpoint Articulate operational scenarios, processes, activities & requirements Services Viewpoint Articulate the performers, activities, services, and their exchanges providing for, or supporting, DoD functions Systems Viewpoint Articulate the legacy systems or independent systems, their composition, interconnectivity, and context providing for, or supporting, DoD functions Project Viewpoint Describes the relationships between operational and capability requirements and the various projects being implemented; Details dependencies between capability management and the Defense Acquisition System process. Figure 3.4.2-1: Architecture Viewpoints in DoDAF V2.0	DoDAF 2.0 volume 1: Introduction, Overview, and Concepts, Manager's Guide, 28 May 2009 available at: https://www.us.army.mil/suite/page/454707
As-built	Final software products (e.g., software requirements, architecture, design documentation) that are consistent with the final implementation are called "as-built."	G. Whittaker

Term	Definition	Source
As-built Documentation	As-built software products often differ from their original design and specifications due to discoveries and refinements that were made during the course of development. As-built documentation updates the original documentation to correctly represent the as-built product.	G. Whittaker
As-run Procedures	Test procedures often need to be red-lined during test dry runs and during run for record. As-run procedures reflect the actual procedures run vice the originally developed procedures.	G. Whittaker
Assessment	An evaluation of processes, products, or resources against a defined set of criteria.	MAIW SW Team
Assurance	Justified confidence that a system will function as intended in its environment of use.	Assurance Cases for Medical Devices 2011, C. Weinstock, SEI
Assurance Case (1 & 2)	(1) A documented body of evidence that provides a convincing and valid argument that a specified set of critical claims regarding a system's properties are adequately justified for a given application in a given environment. (2) An assurance case is a means to provide grounds for confidences and to aid decision making. The assurance case has one or more top-level claims in which confidence is needed and has supporting arguments connecting the top-level claims with multiple levels of sub-claims. The sub-claims are in turn supported by evidence and where appropriate, assumptions.	(1) Structured Assurance Cases: Three Common Standards, T. Scott Ankrum and Alfred H. Kromholz, The MITRE Corporation, October 30, 2009. (2) IEEE P15026-1/D1 Draft Trial-Use Standard for Adoption of ISO/IEC TR 15026-1:2010, Systems and Software Engineering - Systems and Software Assurance - Part 1: Concepts and Vocabulary.

Term	Definition	Source
Assurance Case (3)	(3) General rules for IEEE 15026 compliance are: • The project shall establish and maintain an assurance case. • The project shall ensure that: – Goals and objectives for safety, security, dependability and any other designated critical properties are formulated. – Product assurance-related objectives, properties, or characteristics are explicitly selected for special attention and application of this standard to address the goals and objectives. – Requirements for the achievement of these objectives, properties, or characteristics are defined. – Measures for the requirements are selected and related to the desired characteristics. – Criteria for the achievement or degree or achievement of these objectives, properties, or characteristics are selected and traced to requirements. – Approaches for achieving the objectives, properties, or characteristics are planned, designed, and implemented, as well as demonstrating and documenting that achievement. – The extent of achievement is continuously monitored, documented, and communicated to stakeholders and managers. – An assurance case documenting and communicating the extent of achievement is specified, developed, and maintained as an element of the system. – The artifacts for documenting, analyzing, and communicating the required or claimed properties and characteristics and the extent of achievement are specified, developed, and maintained. – Requirements of the approval authority are satisfied and necessary licenses or certifications are received. As defined in IEEE 15026, scenarios describing overall capability, dependability, safety, security, etc.	(3) Proposed Revision of ISO/IEC 15026: Status Report to MITRE TEM, IEEE CS Liaison Representative to ISO/IEC JTC 1/SC 7, Jim Moore, August 2007.

Term	Definition	Source
ATAM (Architecture Tradeoff Analysis Method)	The SEI Architecture Tradeoff Analysis Method (ATAM) is the leading method in the area of software architecture evaluation. An evaluation using the ATAM typically takes three to four days and gathers together a trained evaluation team, architects, and representatives of the architecture's various stakeholders. Proven benefits of the ATAM include: • Clarified quality attribute requirements • Improved architecture documentation • Documented basis for architectural decisions • Identified risks early in the life cycle • Increased communication among stakeholders Business drivers and the software architecture are elicited from project decision makers. These are refined into scenarios and the architectural decisions made in support of each one. Analysis of scenarios and decisions results in identification of risks, non-risks, sensitivity points, and tradeoff points in the architecture. Risks are synthesized into a set of risk themes, showing how each one threatens a business driver. The most important results are improved architectures. The output of an ATAM is an out-brief presentation and/or a written report that includes the major findings of the evaluation. These are typically: • The architectural styles identified • A "utility tree" — a hierarchic model of the driving architectural requirements • The set of scenarios generated and the subset that were mapped onto the architecture • A set of quality-attribute-specific questions that were applied to the architecture and the responses to these questions • A set of identified risks • A set of identified non-risks	ATAM Architecture Tradeoff Analysis Method, (http://www.sei.cmu.edu/architecture/tools/atam/)

Term	Definition	Source
ATAM Diagram	This diagram accompanies the glossary entry for ATAM  Business Drivers Quality Attributes Scenarios Analysis Architectural Plan Architectural Approaches Architectural Decisions Tradeoffs Impacts Sensitivity Points Non-Risks Distilled into Risk Themes Risks	(http://www.sei.cmu.edu/architecture/tools/atam/)
Audit	(1) An independent examination of a software product, software process, or set of software processes to assess compliance with specifications, standards, contractual agreements, or other criteria. (2) Systematic, independent, and documented process for obtaining audit evidence and evaluating it objectively to determine the extent to which audit criteria are fulfilled. Note: An audit should result in a clear indication of whether the audit criteria have been met.	(1) (IEEE 1028-2008 IEEE Standard for Software Reviews and Audits, 3.2) (2) (ISO/IEC 15288:2008 Systems and software engineering--System life cycle processes, 4.6)

Term	Definition	Source
Behavioral	Behavioral vs. static analysis examines software in action vs. the software code. Behavior entails transitioning states and modes, executing rules, work flows, dynamic performance, resource consumption and management, etc.	G. Whittaker
Bi-directional Traceability	Bi-directional traceability between sets A and B, (e.g., requirements, architectural elements, test procedures) means that the following mappings exist: from each member of A to one or more members of B; and from each member of B to one or more members of A.	G. Whittaker
Build Scripts	A set of reusable procedures, expressed in a scripting language interpretable by a tool such as Make or ANT to automate the building of software products. They are typically used to encapsulate complex build instructions and to enforce systematic and correct building of executables or dynamic load libraries, shared libraries, etc.	G. Whittaker
Coding	(1) In software engineering, the process of expressing a computer program in a programming language. (2) The transforming of logic and data from design specifications (design descriptions) into a programming language.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)

Term	Definition	Source
Communication Pattern	Communication patterns are very important to the design effort and exist at many levels of abstraction; three key levels are described here: (1) At the symbol and information structure level, communication patterns are design patterns for input/output protocols and may specify specific symbol or bit sequences to be used for signifying the demarcation of information or data elements composing a message structure or communication protocol. (2) At an architectural level, communication patterns refer to critical information or data flow paths revealed by analysis of command, control, and telemetry scenarios. Executable architectures are used to model the high level behavior that exhibits these patterns so that potential performance bottlenecks or single points of failure are eliminated in the early design phase. (3) During detailed software design, modules, components and functions are defined to realize the software architectural elements (and emulate or simulate hardware elements) to show how they will satisfy the imposed requirements. At this level communication patterns between software modules are typically described in UML sequence diagrams that illustrate the temporal ordering and sequential dependency of inter-module calls necessary to satisfy a particular use-case or function. Sequence diagrams may be created for a number of scenarios driven by DITL or TLYF or other end-to-end stress testing requirements. Key performance, reliability and information assurance requirements come into play at this level and frequently drive competing design concerns.	(1 - 3) G. Whittaker
Completeness	(1) In some cases completeness is a measureable objective quantity indicating the degree to which a set of actions have been executed and finished (e.g., the completeness of test results or the completeness of a particular software build). (2) In other cases the completeness is taken as a subjective judgment regarding the adequacy of an information product (e.g., requirements, architecture, design) to capture the developer's and acquirer's needs. For example, completeness in test-procedures indicates the adequacy of the procedures to cover nominal and off-nominal excursions - these are almost never complete in the former sense due to the combinatorial nature of input cases required, but a large set of representative cases may be necessary to establish a spanning range of input and environmental conditions when stress testing a critical software component. Such testing may be said to exhibit "completeness" but it is really only a subjective assessment based on software testing experience.	(1-2) G. Whittaker

Term	Definition	Source
Conceptual Data Model	A data model that illustrates the data groups as they are seen by the user.	(ISO/IEC 24570:2005 Software engineering -- NESMA functional size measurement method version 2.1 -- Definitions and counting guidelines for the application of Function Point Analysis)
Configuration Identification	(1) An element of configuration management, consisting of selecting the configuration items for a system and recording their functional and physical characteristics in technical documentation. (2) The current approved technical documentation for a configuration item as set forth in specifications, drawings, associated lists, and documents referenced therein.	(1 - 2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Configuration Item	(1) Entity within a configuration that satisfies an end use function and that can be uniquely identified at a given reference point. (2) Item or aggregation of hardware, software, or both that is designed to be managed as a single entity. (3) Component of an infrastructure or an item which is, or will be, under the control of configuration management. (4) An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process. (5) Aggregation of work products that is designated for configuration management and treated as a single entity in the configuration management process. Note: Configuration items may vary widely in complexity, size and type, ranging from an entire system including all hardware, software and documentation, to a single module or a minor hardware component.	(1) (ISO/IEC 12207:2008 Systems and software engineering--Software life cycle processes, 4.7) (2) (ISO/IEC 19770-1:2006 Information technology -- Software asset management -- Part 1: Processes, 3. 2) (3) (ISO/IEC 20000-1:2005 Information technology -- Service management -- Part 1: Specification, 2.4) (4) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (5) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (Note) http://pascal.computer.org/sev_display/index.action
Configuration Item List	A table or listing of configuration items.	G. Whittaker

Term	Definition	Source
Configuration Management	(1) A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements. (2) Technical and organizational activities comprising configuration identification, control, status accounting, and auditing.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) (ISO/IEC 29881:2008 Information technology--Software and systems engineering--FiSMA 1.1 functional size measurement method, 4.9)
Contingency Plan	A plan for dealing with a risk factor should it become a problem.	(ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Controlled	Controlled products are governed by a configuration management process, but not necessarily required to be approved by a configuration control board. In other words a check-in, check-out, and version management capability is used to ensure that controlled documents have official versions which are recoverable during their life cycle.	G. Whittaker
COTS	(1) A product available for purchase and use without the need to conduct development activities. (2) An item that a supplier offers to several acquirers for general use. Note: COTS software product includes the product description (including all cover information, data sheet, web site information, etc.), the user documentation (necessary to install and use the software), the software contained on a computer sensible media (disk, CD-ROM, internet downloadable, etc.). Software is mainly composed of programs and data. This definition applies also to product descriptions, user documentation and software which are produced and supported as separate manufactured goods, but for which typical commercial fees and licensing considerations may not apply. See Also: software product.	(1) (ISO/IEC 90003:2004 Software engineering -- Guidelines for the application of ISO 9001:2000 to computer software, 3.4) (2) (ISO/IEC 15289:2006 Systems and software engineering--Contents of systems and software life cycle information products (Documentation), 5.2)
COTS Selection	The process of selecting COTS or the results of such a selection process. The selection process should follow a set of rational criteria and assumptions clearly documented including any trade-off analysis (e.g., cost vs. benefits) that was conducted to arrive at the final COTS selection.	G. Whittaker
Criticality	The degree of impact that a requirement, module, error, fault, failure, or other characteristic has on the development or operation of a system.	(IEEE 829-2008 IEEE Standard for Software and System Test Documentation, 3.1.9)

Term	Definition	Source
Criticality Analysis	A procedure by which each potential failure mode is ranked according to the combined influence of severity and probability of occurrence. Criticality Analyses are typically performed as part of a Failure Mode Effects Criticality Analysis (FMECA). Per MIL-STD-1629A Criticality Analysis (task 102) follows the Failure Mode and Effects Analysis (FMEA, task 101).	MIL-STD-1629A
Criticality Analysis Report	The results of the FMEA and other related analyses shall be documented in a report that identifies the level of analysis, summarizes the results, documents the data sources and techniques used in performing the analysis, and includes the system definition narrative, resultant analysis data, and worksheets. The worksheets shall be organized to first display the highest indenture level of analysis and then proceed down through decreasing indenture levels of the system. The ground rules, analysis assumptions, and block diagrams shall be included, as applicable, for each indenture level analyzed. Interim reports shall be available at each design review to provide comparisons of alternative designs and to highlight the Category I and Category II failure modes, the potential single failure points, and the proposed design corrections. The final report shall reflect the final design and provide identification of the Category I and Category II failure modes and the single failure points which could not be eliminated from the design.	MIL-STD-1629A
Data Model	(1) A graphical and textual representation of analysis that identifies the data needed to achieve system mission, functions, goals, objectives, and strategies. (2) A model about data by which an interpretation of the data can be obtained in a modeling tool. Note: A data model is one that may be encoded and manipulated by a computer. A data model identifies the entities, domains (attributes), and relationships (associations) with other data and provides the conceptual view of the data and the relationships among data.	(1) (IEEE 1320.2-1998 (R2004) IEEE Standard for Conceptual Modeling Language Syntax and Semantics for IDEF1X97 (IDEFobject), 3.1.44) (2) (ISO/IEC 15474-1:2002 Information technology -- CDIF framework -- Part 1: Overview, 4.2)

Term	Definition	Source
Database	(1) A collection of interrelated data stored together in one or more computerized files. (2) A collection of data organized according to a conceptual structure describing the characteristics of the data and the relationships among their corresponding entities, supporting one or more application areas. (3) A collection of data describing a specific target area that is used and updated by one or more applications.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) (ISO/IEC 2382-1:1993 Information technology--Vocabulary--Part 1: Fundamental terms, 01.08.05) (3) (ISO/IEC 29881:2008 Information technology--Software and systems engineering--FiSMA 1.1 functional size measurement method, A.5)
Day in the Life Testing	A scenario based on or composed of a sequence of operational activities that must be supported by the objective system that spans a significant range of interdependent capabilities and services provided by the system.	G. Whittaker
Demonstration	A dynamic analysis technique that relies on observation of system or component behavior during execution, without need for post-execution analysis, to detect errors, violations of development standards, and other problems. One of the four types of verification methods: demonstration, inspection, test, and analysis.	(ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Dependability	(1) Dependability is defined as the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers. The service delivered by a system is its behavior as it is perceived by its user(s); a user is another system (human or physical) which interacts with the former. (2) Dependability encompasses reliability, maintainability, and availability.	(1) From Laprie, J.C., Editor. *Dependability: Basic Concepts and Terminology*. Vienna, Austria: Springer-Verlag, 1992, as cited in Technical Note CMU/SEI-2004-TN-016, *Dependability Cases - Performance Critical Systems*, Weinstock, Charles. B., Goodenough, John B., and Hudak, John J. (2) M. Hecht

Term	Definition	Source
Design	(1) The process of defining the architecture, components, interfaces, and other characteristics of a system or component. (2) The result of the process in (1). (3) The process of defining the software architecture, components, modules, interfaces, and data for a software system to satisfy specified requirements. (4) The process of conceiving, inventing, or contriving a scheme for turning a computer program specification into an operational program. (5) The activity that links requirements analysis to coding and debugging. (6) The stage of documentation development that is concerned with determining what documentation will be provided in a product and the nature of that documentation.	(1 - 5) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (6) (ISO/IEC 26514:2008 Systems and software engineering--requirements for designers and developers of user documentation, 4.13)
Design Patterns	A description of the problem and the essence of its solution to enable the solution to be reused in different settings. Note: not a detailed specification, but a description of accumulated wisdom and experience.	(ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Deviation	(1) A departure from a specified requirement. (2) A written authorization, granted prior to the manufacture of an item, to depart from a particular performance or design requirement for a specific number of units or a specific period of time. (3) A departure from a released plan or controlled process.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (3) MAIW SW committee
Diagnostic	Pertaining to the detection and isolation of faults or failures (e.g., a diagnostic message, a diagnostic manual).	(ISO/IEC 24765:2009 Systems and software engineering vocabulary)

Term	Definition	Source
Document	(1) A uniquely identified unit of information for human use, such as a report, specification, manual or book, in printed or electronic form. (2) To create a document as in (1). (3) To add comments to a computer program. (4) An item of documentation. (5) A medium and the information recorded on it, which generally has permanence and can be read by a person or a machine. (6) Information and its supporting medium. (7) A separately identified piece of documentation which could be part of a documentation set, (e.g., In software engineering: project plans, specifications, test plans, user manuals). Note: Documents include both paper and electronic documents.	(1) (ISO/IEC TR 9294:2005 Information technology -- Guidelines for the management of software documentation, 3.1) (2) (IEEE 829-2008 IEEE Standard for Software and System Test Documentation, 3.1.11) (3) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (4) (ISO/IEC 15910:1999 Information technology -- Software user documentation process, 4.1) (5) (IEEE 829-2008 IEEE Standard for Software and System Test Documentation, 3.1.11) (6) (ISO/IEC 20000-1:2005 Information technology -- Service management -- Part 1: Specification, 2.6) (7) (ISO/IEC 26514:2008 Systems and software engineering-- requirements for designers and developers of user documentation, 4.15)
Earned Value Management (EVM)	A management methodology for integrating scope, schedule, and resources, and for objectively measuring project performance and progress. Performance is measured by determining the budgeted cost of work performed (i.e., earned value) and comparing it to the actual cost of work performed (i.e., actual cost).	(A Guide to the Project Management Body of Knowledge (PMBOK® Guide) -- Fourth Edition)

Term	Definition	Source
Emulation	(1) A model that accepts the same inputs and produces the same outputs as a given system. (2) The process of developing or using a model. (3) The use of a data processing system to imitate another data processing system, so that the imitating system accepts the same data, executes the same programs, and achieves the same results as the imitated system. See Also: simulation	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (3) (ISO/IEC 2382-1:1993 Information technology--Vocabulary--Part 1: Fundamental terms, 01.06.02)
Dependability	The ability to deliver service that can justifiably be trusted. This definition stresses the need for justification of trust. As developed over the past three decades, dependability is an integrating concept that encompasses the following attributes: availability--readiness for correct service; reliability--continuity of correct service; safety--absence of catastrophic consequences on the user(s) and the environment; confidentiality--absence of unauthorized disclosure of information; integrity--absence of improper system alterations; maintainability--ability to undergo, modifications, and repairs. The dependability specification of a system must include the requirements for the dependability attributes in terms of the acceptable frequency and severity of failures for the specified classes of faults and a given use environment. One or more attributes may not be required at all for a given system.	Dependability and its threats: A Taxonomy, Algirdas Avizˇienis, Jean-Claude Laprie, Brian Randell
End-to-End Testing	End-to-end testing typically utilizes external interfaces to stimulate system activity that is representative of typical usage (e.g., DITL, TLYF) as well as stress testing critical communication patterns throughout the software and hardware system implementation. Some of these tests may require specialized instrumentation to measure internal parameters during the course of the testing or to introduce spurious signals, noise, faults, or other information on internal interfaces to represent environmental effects or other stressing phenomena.	G. Whittaker

Term	Definition	Source
Error Handling	Error handling refers to the anticipation, detection, and resolution of programming, application, and communications errors. Specialized software routines, called error handlers, are available for some applications. The best routines of this type forestall errors if possible, recover from them when they occur without terminating the application, or (if all else fails) gracefully terminate an affected application and save the error information to a log file.	http://searchsoftwarequality.techtarget.com/definition/error-handling
Executable Architecture (EA)	(1) An Executable Architecture, in general, is the description of a system architecture (including software and/or otherwise) in a formal notation, together with the tools (e.g., compilers/translators) that allow the automatic or semi-automatic generation of artifacts (e.g., Capability Gap Analysis [CGA], models, software stubs, Military Scenario Definition Language [MSDL]) from that notation and which are used in the analysis, refinement, and/or the implementation of the architecture described. (2) A dynamic model of sequenced activities/events (concurrent or sequential) performed at an operational node by roles (within organizations) using resources (systems) to produce and consume information and data. (3) A modeling and simulation approach that enables event driven or time stepped activation of modes and states of the modeled system. Different system elements (hardware or software) of an executable architecture may be elaborated within the model to differing levels of detail to analyze specific communication patterns and their effects on overall model performance.	(1) Wikipedia (2) Executable Architecture Methodology for Analysis, FY04 Final Report (Pawlowski III, et al., 2004) (3) G. Whittaker
eXtreme Programming	One of several agile methodologies that incrementally release capabilities to the customer. See the cited reference for a complete definition of the features of eXtreme programming.	G. Whittaker For details see: Kent Beck </wiki/Kent_Beck> or Cynthia Andres. “Extreme Programming Explained: Embrace Change,” Second Edition, Addison-Wesley.
Fault	(1) A manifestation of an error in software. (2) An incorrect step, process, or data definition in a computer program. (3) A defect in a hardware device or component. Note: A fault, if encountered, may cause a failure. Syn: bug.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2 -3) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)

Term	Definition	Source
Fault Management	System functional element(s) that collectively provide system fault tolerance. Fault management functionality includes fault detection, fault isolation, fault repair, service restoration or transition to safe mode. See also: fault tolerance.	G. Whittaker
Fault Tolerance	(1) The ability of a system or component to continue normal operation despite the presence of hardware or software faults. (2) The number of faults a system or component can withstand before normal operation is impaired. (3) Pertaining to the study of errors, faults, and failures, and of methods for enabling systems to continue normal operation in the presence of faults. Syn: error tolerance; fail safe; fail soft; fault secure; robustness.	(1 - 3) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Hardware Compatibility (1&2)	(1) Compiled code is said to be *compatible with a specific target (hardware)* processor and memory architecture when it can be loaded and executed on the target. This means that the machine architecture (von Neumann, Quantum, etc.) instructions, data representation and structure, word size, byte size and bit order (endianess) within a byte on the target hardware (processor and storage devices) and the buses between the processor, instruction and data stores are what the compiler designers expected. In practice it also usually means that a target hardware operating system (OS) exists and is *compatible* with the compiler in that its calls to kernel OS services, interrupts and modes of operation are supported. (2) In order to avoid the von Neumann bottleneck (instruction and data bus limitations) high performance real-time algorithms are often only *hardware compatible* with a specialized processor architecture (e.g., non-von Neuman variations of the memory models and characteristics such as content addressable memory (CAM), application specific integrated circuits (ASICS) supporting specialized cache strategies and instruction pipelines, field programmable gate arrays (FPGAs) or parallel computing models such as single instruction multiple data -SIMD, multiple instruction multiple data - MIMD, dataflow architecture, etc.)	(1 - 2) G. Whittaker

Term	Definition	Source
Hardware Compatibility (3&4)	(3) General purpose software application development *hardware compatibility* is based on selected programming language compiler and OS availability, satisfactory processor speed, required standard peripherals and Input/output (IO) device support, graphics display performance, and perhaps some lower level interrupt handling, throughput and memory resource capacity requirements. (4) Operational system level *hardware compatibility* means that the software architectural and design elements and principles are satisfied or satisfiable (e.g., client server, n-tier web service, net-centric, grid or cloud computing, and supporting messaging protocols are or can be supported by the configuration of processor(s), OS, firmware, device drivers, buses, routers, switches, etc. comprising the target system hardware).	(3 - 4) G. Whittaker
Hardware in the Loop Simulation (HIL or HITL)	Hardware-in-the-loop (HIL) simulation is a technique that is used in the development and test of complex real-time embedded systems. HIL simulation provides an effective platform by adding the complexity of the system under control to the test platform. The complexity of the system under control is included in test and development by adding a mathematical representation of all related dynamic systems. These mathematical representations are referred to as the “system simulation.” The embedded system to be tested interacts with this simulation.	Wikipedia

Term	Definition	Source
Implementation	(1) The process of translating a design into hardware components, software components, or both. (2) The result of the process in (1). (3) A definition that provides the information needed to create an object and allow the object to participate in providing an appropriate set of services. (4) The installation and customization of packaged software. (5) Construction. (6) The system development phase at the end of which the hardware, software, and procedures of the system considered become operational. (7) A process of instantiation whose validity can be subject to test. (8) Phase of development during which user documentation is created according to the design, tested, and revised.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary (2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (3) (ISO/IEC 19500-2:2003 Information technology -- Open Distributed Processing -- Part 2: General Inter-ORB Protocol (GIOP)/Internet Inter-ORB Protocol (IIOP), 3.2.8) (4) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (5) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (6) (ISO/IEC 2382-20:1990 Information technology--Vocabulary--Part 20: System development, 20.04.01) (7) (ISO/IEC 10746-3:1996 Information technology -- Open Distributed Processing -- Reference Model: Architecture, 9.1.2) (8) (ISO/IEC 26514:2008 Systems and software engineering--requirements for designers and developers of user documentation, 4.24)

Term	Definition	Source
Information Assurance	Information assurance (IA) is the practice of managing risks related to the use, processing, storage, and transmission of information or data and the systems and processes used for those purposes. While focused dominantly on information in digital form, the full range of IA encompasses not only digital but also analog or physical form. Information assurance as a field has grown from the practice of information security which in turn grew out of practices and procedures of computer security. IA includes physical security, system security, anti-tamper, and privacy, and addresses network security and data integrity.	Wikipedia
Information Assurance Plan	An IA plan identifies the information assurance model, requirements, standards, and practices to be applied and integrated into the software intensive system throughout its full life-cycle. The plan should identify those architectural, operational, and supporting infrastructure elements that provide key support for the IA strategy. The plan should ensure that the designated certification and accreditation agency and liaisons are identified and that the acquisition IA SMEs are integrated into the system and software coops and design development from the beginning of conceptual design and all the way through to final accreditation and authorization to operate.	G. Whittaker
Installation Instructions	Documented detailed sequential steps and procedures that explain exactly how to install a particular software product or set of products on the host system.	G. Whittaker
Integration Testing	Testing conducted where software components, hardware components, or both are combined to evaluate the correctness of the interactions among them in accordance with the interface documentation. Note: commonly used for both the integration of components and the integration of entire systems.	Adapted from: (IEEE 1012-2004 IEEE Standard for Software Verification and Validation, 3.1.14), (IEEE 829-2008 IEEE Standard for Software and System Test Documentation, 3.1.14)

Term	Definition	Source
Interface Design Document (IDD)	(1) A description of the architecture and design of interfaces between systems or among the components of a system. These descriptions include control algorithms, protocols, data contents and formats, and performance. See Also: interface requirements specification (IRS). Sometimes this information is contained in an interface control document. (2) The interface design description (IDD) describes the interface characteristics of one or more systems, subsystems, Hardware Configuration Items (HWCSI), Computer Software Configuration Items (CSCIs), manual operations, or other system components. An IDD may describe any number of interfaces. The IDD can be used to supplement the System/Subsystem Design Description (SSDD) (DI-IPSC-81432A), Software Design Description (SDD) (DI-IPC-81435A), and Database Design Description (DBDD) (DI-IPSC-81437A). The IDD and its companion Interface Requirements Specification (IRS) (DI-IPSC-81434A) serve to communicate and control interface design decisions.	(1) Adapted from (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) Interface Design Description (IDD) DI-IPSC-81436A
Interface Requirements Specification (IRS)	(1) Documentation that specifies requirements for interfaces between systems or among the components of a system. These requirements may include constraints on formats and timing. (2) The Interface Requirements Specification (IRS) specifies the requirements imposed on one or more systems, subsystems, Hardware Configuration Items (HWCIs), Computer Software Configuration Items (CSCIs), manual operations, or other system components to achieve one or more interfaces among these entities. An IRS can cover any number of interfaces. The IRS can be used to supplement the System/Subsystem Specification (SSS), (DI-IPSC-81431A) and Software Requirements Specification (SRS) (DI-IPSC-81433A) as the basis for design and qualification of the testing of systems and CSCIs.	(1) Adapted from: (IEEE 1012-2004 IEEE Standard for Software Verification and Validation, 3.1.17), (IEEE 829-2008 IEEE Standard for Software and System Test Documentation, 3.1.17) (2) Interface Requirements Specification (IRS) DI-IPSC-81434A
Interface Testing	Testing conducted to evaluate whether systems or components pass data and control correctly to one another in accordance with the interface documentation. See Also: component testing, integration testing, system testing, unit test.	Adapted from (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Lessons Learned	The learning gained from the process of performing the project. Lessons learned may be identified at any point and included in a lessons learned knowledge base.	Adapted from (A Guide to the Project Management Body of Knowledge (PMBOK® Guide) -- Fourth Edition)

Term	Definition	Source
Lessons Learned Knowledge base	A store of historical information and lessons learned about both the outcomes of previous project selection decisions and previous project performance.	(A Guide to the Project Management Body of Knowledge (PMBOK® Guide) -- Fourth Edition)
Life Cycle Model	Framework of processes and activities concerned with the development, operations or sustainment of software or systems that may be organized into stages, which also acts as a common reference for communication and understanding.	(ISO/IEC 12207:2008 Systems and software engineering--Software life cycle processes, 4.12 & 4.16)
Log	(1) Log - a document used to record and describe or denote selected items identified during execution of a process or activity. Usually used with a modifier, such as issue, quality control, action, or defect. (2) Test log - chronological record of relevant details about the execution of tests	(1) (A Guide to the Project Management Body of Knowledge (PMBOK® Guide) -- Fourth Edition) (2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Logical data model (LDM)	A logical data model in systems engineering is a representation of an organization's data, organized in terms of entities and relationships, and is independent of any particular data management technology.	Wikipedia
Maintenance	Software maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes.	Wikipedia
Maintenance Environment	Facilities, hardware, software, tools, training and documentation that are required in order to maintain a particular set of software.	Adapted from G. Whittaker
Mapping	A mapping is an assigned correspondence or relationship between two well defined families or sets (technically identified as a domain and a co-domain) such that for each element of the domain one and only one element of the co-domain is designated by the mapping. For example, a set of system requirements may be mapped to sets of derived requirements, or sets of architectural elements or sets of software requirements. The domain in each of these cases is the set of system requirements and the co-domain in each of these cases is the set of subsets (power set) of the derived requirements, architectural elements, or software requirements. The key is that the mapping designates only one member of the co-domain for each element in the domain but there may be many members in the domain that map to the same element in the co-domain. See Bi-directional traceability and traceability.	G. Whittaker

Term	Definition	Source
Master Software Build Plan or Master Software Integration Plan	A plan for the implementation and integration of software components into builds or increments. The plan must cover the software requirements allocated to each build or increment and describes the sequence of integration of the software units that comprise the build or increment.	G. Whittaker
Mitigation Plan	A plan for actions to reduce the likelihood of a risk occurring or to reduce the consequences should it become a problem.	S. Eslinger
Modeling, Simulation, and Analyses (MS&A)	Modeling and simulation (M&S) or modeling, simulation and analysis (MS&A) generally refers to computational models that have been created to explore and analyze dynamic system behaviors. MS&A applied to system and software architecture is known as Executable Architecture. Other software related examples include: reliability, criticality and safety modeling and analyses.	G. Whittaker
NDI analysis	Non-developmental item analysis is performed in support of trade-off analysis between COTS, GOTS and custom developed items to decide on the best strategy for the overall program's requirements and constraints.	G. Whittaker
Non-Developmental Item	The term non-developmental is used to describe items that were previously developed. For example, for software, a non-developmental item may be COTS, GOTS, legacy re-use, or open source.	MAIW SW Team
Non-functional Requirements	A requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors, (e.g., requirements for design constraints, adaptability, quality, RMA).	MAIW SW Team
Objective Evidence	Data supporting the existence or verity of something. Note: Objective evidence may be obtained through observation, measurement, test, interviews, document reviews and other means.	(Adapted from ISO/IEC 15504-1:2004 Information technology -- Process assessment -- Part 1: Concepts and vocabulary, 3.24)
Objective(s)	(1) Something toward which work is to be directed, a strategic position to be attained, or a purpose to be achieved, a result to be obtained, a product to be produced, or a service to be performed. Syn: Purpose	(A Guide to the Project Management Body of Knowledge (PMBOK® Guide) -- Fourth Edition)

Term	Definition	Source
Operator(s)	Entity that performs the operation(s) of a system	(ISO/IEC 12207:2008 Systems and software engineering--Software life cycle processes, 4.22) (ISO/IEC 15288:2008 Systems and software engineering--System life cycle processes, 4.13) (ISO/IEC 15939:2007 Systems and software engineering--Measurement process, 3.30)
Pass/Fail Criteria	Decision rules used to determine whether a software item or a software feature passes or fails a test.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Peer Review	As used in this document, a peer review is a type of product evaluation characterized by: a) action items are documented and tracked to closure; b) defects are documented, categorized by type and severity, and tracked to closure; c) reviewers are peers of the author of the product being evaluated.	MAIW SW Team
Physical Data Model	A physical data model (a.k.a. database design) is a representation of a data design which takes into account the facilities and constraints of a given database management system. In the lifecycle of a project, it is typically derived from a logical data model, though it may be reverse-engineered from a given database implementation. A complete physical data model will include all the database artifacts required to create relationships between tables or achieve performance goals, such as indexes, constraint definitions, linking tables, partitioned tables or clusters. The physical data model can usually be used to calculate storage estimates and may include specific storage allocation details for a given database system.	Wikipedia
Plan	A documented form of planning information products - not necessarily a formal document. See also: Project Plan, Software Development Plan, Test Plan	G. Whittaker
Planning Information	Information necessary to support the process of creating a plan. The planning information may include identification of goals and objectives, constraints, processes, and activities that are to be organized, allocated resources and sequenced to achieve the planning goals and objectives.	G. Whittaker

Term	Definition	Source
Procedure	Defines in detail when and how to perform certain activities or tasks, including tools needed. A procedure typically includes the following elements: a) Date of issue and status b) Scope c) Issuing organization d) Approval authority e) Roles and responsibilities f) Relationship to policies, plans and other procedures g) Authoritative references h) Inputs and outputs i) Ordered description of steps to be taken by each participant j) Error and problem resolution k) Glossary l) Change history	Adapted from ISO/IEC/IEEE 15288:2008, 5.3.1
Process	Set of interrelated or interacting activities which transforms inputs into outputs.	(ISO/IEC 15288:2008 Systems and software engineering—System life cycle processes, 4.16) (ISO/IEC 15939:2007 Systems and software engineering--Measurement process, 3.32)
Product Evaluation	An evaluation of a product by knowledgeable reviewers and relevant stakeholders, other than the author of the product, against a defined set of criteria. For the purpose of this document, a product evaluation is a general term for assessments with different levels of formality (e.g., peer reviews, inspections, or walkthroughs).	
Program Plan	A document that describes the technical and management approach to be followed for a program. Note: For example: a software development plan. The plan typically describes the work to be done, the resources required, the methods to be used, the procedures to be followed, the schedules to be met, and the way that the program will be organized.	Adapted from (ISO/IEC 24765:2009 Systems and software engineering vocabulary)

Term	Definition	Source
Prototype	(1) A preliminary type, form, or instance of a system that serves as a model for later stages or for the final, complete version of the system. (2) Model or preliminary implementation of a piece of software suitable for the evaluation of system design, performance, or production potential, or for the better understanding of the software requirements. Note: A prototype may be used to get feedback from users for improving and specifying a complex human interface, for feasibility studies, or for identifying requirements.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) (ISO/IEC 15910:1999 Information technology—Software user documentation process, 4.41)
Prototyping	A hardware and software development technique in which a preliminary version of part or all of the hardware or software is developed to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process.	(ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Quality Attribute	(1) Characteristic of software, or a generic term applying to quality factors, quality subfactors, or metric values. (2) Requirement that specifies the degree of an attribute that affects the quality that the system or software must possess. See also ATAM.	(1) (IEEE 1061-1998 (R2004) IEEE Standard for Software Quality Metrics Methodology, 2.17) (2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Regression Test(ing)	(1) Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements. (2) Testing required to determine that a change to a system component has not adversely affected functionality, reliability or performance and has not introduced additional defects.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) (ISO/IEC 90003:2004 Software engineering—Guidelines for the application of ISO 9001:2000 to computer software, 3.11)
Released	Released product, or product documentation, as used in this guide, is controlled by a configuration management process and in addition has been approved for *release* by a configuration control board or other official board review (e.g., Engineering Review Board, Architecture Control Board).	G. Whittaker

Term	Definition	Source
Repeatability	(1) The same stimuli to a system in given initial conditions gives the same expected results. Test repeatability is a key driving factor behind all agile, test driven or extreme development methods. (2) Test repeatability is an attribute of a test, indicating that the same results are produced each time the test is conducted.	(1) Adapted from http://www.cydone.com/node/42 (2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Reprogrammability	The ability to modify the contents of memory associated with an embedded processor (e.g., on-board processors). This Includes modification of computer instructions, data, or replacement of the entire memory content.	MAIW SW Team
Requirements Allocation	The assignment or budgeting of top-level requirements among the lower-level partitioned elements or sub-systems. Note: In this manner, the system elements that perform all or part of specific requirements are identified.	Adapted from (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Requirements Allocation Matrix	A structure that documents the bi-directional mapping between the software items in the system architecture and the system requirements allocated to software.	G. Whittaker
Requirements Traceability	Requirements are documented so that each requirement in a higher-level or ‘parent’ requirements specification is mapped to one or more ‘child’ requirements in associated lower-level requirements specifications. In addition, all requirements in the associated lower-level or ‘child’ requirements specifications are mapped to one or more parent requirement(s) in the higher level requirement specification. This type of mapping is called bi-directional traceability.	R. Wilkes
Requirements Verification	See Verification.	
Retest	Retest is conducted after corrections have been made in response to a prior test failure.	G. Whittaker
Reuse	The use of existing software or software knowledge to build new software.	Wikipedia
Risk Handling	A course of action taken in response to a risk factor. Note: includes risk acceptance, risk avoidance, risk transfer, and risk mitigation.	(ISO/IEC 24765:2009 Systems and software engineering vocabulary)

Term	Definition	Source
Risk Mitigation	(1) A course of action taken to reduce the probability of and potential loss from a risk factor. (2) [Technique] a risk response planning technique associated with threats that seeks to reduce the probability of occurrence or impact of a risk to below an acceptable threshold. Note: includes executing contingency plans when a risk metric crosses a predetermined threshold (when a risk factor becomes a problem)	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) (A Guide to the Project Management Body of Knowledge (PMBOK® Guide)—Fourth Edition)
Root Cause Analysis	(1) An analytical technique used to determine the basic underlying reason that causes a variance, defect, or risk. (2) Determination of a potential problem's underlying cause or causes.	(1) (A Guide to the Project Management Body of Knowledge (PMBOK® Guide)—Fourth Edition) (2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Scrum	An iterative, incremental framework for project management often seen in agile software development.	Wikipedia
Simulation	(1) A model that behaves or operates like a given system when provided a set of controlled inputs. (2) The process of developing or using a model as in (1). (3) The use of a data processing system to represent selected behavioral characteristics of a physical or abstract system.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (3) (ISO/IEC 2382-1:1993 Information technology—Vocabulary—Part 1: Fundamental terms, 01.06.01)
Software Architecture	The software architecture of a program or computer system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.	[Extracted from Use of Quality Attribute Workshops (QAWs) in Source Selection for a DoD System Acquisition: A Case Study, John K. Bergey and William G. Wood, June 2002]

Term	Definition	Source
Software Architecture Description	The software architecture of the system is the set of structures needed to reason about the system which comprise software elements, the relations among them and the properties of both. Documenting software architecture facilitates communications between stakeholders, documents early decisions about high level design, and allows reuse of design components and patterns between programs.	Wikipedia
Software Architecture Views	Software architecture is commonly organized in views, which are analogous to the different types of blueprints made in building architecture. A view is a representation of a set of system components and relationships among them. Within the ontology established by ANSI/IEEE 1471-2000, views are responses to viewpoints, where a viewpoint is a specification that describes the architecture in question from the perspective of a given set of stakeholders and their concerns. The viewpoint specifies not only the concerns addressed but the presentation, model kinds used, conventions used and any consistency (correspondence) rules to keep a view consistent with other views.	Wikipedia
Software Configuration Management	A discipline applying technical and administrative direction and surveillance over the life cycle of items to: (1) Identify and document the functional and physical characteristics of configuration items. (2) Control changes to configuration items and their related documentation. (3) Record and report information needed to manage configuration items effectively, including the status of proposed changes and implementation status of approved changes. (4) Audit configuration items to verify conformance to specifications, drawings, interface control documents, and other contract requirements.	MIL-STD-973, Configuration Management, 17 April 1992

Term	Definition	Source
Software Design Description	The software design description presents the characteristics of one or more systems, subsystems, software items, or other system components, and their interfaces. It includes the following: a) Identification of external interfaces, software components, software units, and other interfaces. b) Allocation of software item requirements to software components, further refined, as needed, to facilitate detail design. c) Description of the items (systems, configuration items, users, hardware, software, etc.) that must communicate with other items to pass and receive data, instructions or information. d) The concept of execution including data flow and control flow. e) Security considerations. f) Reuse elements. g) Error handling. h) Identification of reference and protocol documents or specifications.	ISO/IEC/IEEE 12207:2008
Software Development Plan	The development plan presents how the organization or program plans to conduct development activities (the software implementation strategy).	ISO/IEC/IEEE 12207:2008
Software Identification	See configuration identification.	
Software Item	(1) Source code, object code, control code, control data, or a collection of these items. (2) An aggregation of software, such as a computer program or database, that satisfies an end use function and is designated for specification, qualification testing, interfacing, configuration management, or other purposes.	(1) (ISO/IEC 12207:2008 Systems and software engineering--Software life cycle processes, 4.41) (2) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Software Life Cycle Model	A software life cycle model is a structure imposed on the development of a software product. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. See Section 2 for software development models discussed in this document.	Adapted from Wikipedia
Software Peer Review(s)	A type of software evaluation in which a work product (document, code, or other) is examined by its author and one or more colleagues, in order to evaluate its technical content and quality.	Wikipedia

Term	Definition	Source
Software Problem	A bug, error, flaw, mistake, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways.	Wikipedia
Software process Evaluation	Software process evaluations are performed for two purposes: 1) To determine the quality of the software development processes of the program. These software process evaluations are performed by a team of software professionals independent of the developers of the processes. 2) To determine the adherence of the software developers to the defined processes. These software development evaluations are performed by a team of software professionals independent of the software developers.	R. Fulford/ L. Holloway
Software Product Evaluation	An evaluation of a software product by persons other than the author with different levels of required formality, documentation, and stakeholders. For example: peer reviews, inspections, or walkthroughs.	R. Fulford
Software Quality Assurance	A planned and systematic approach to the evaluation of the quality of and adherence to software product standards, processes, and procedures. This includes the process of assuring that standards and procedures are established and are followed throughout the software life cycle. Compliance with agreed-upon standards and procedures is evaluated through process monitoring, product evaluation, and audits.	http://satc.gsfc.nasa.gov/assure/agbsec3.txt
Software Team Member	An individual that performs software activities, including developers, testers, software configuration managers, and software quality assurance.	R. Fulford
Software Test Environment	The facilities, hardware, software, firmware, procedures, and documentation needed to perform qualification or other testing of software Note: Elements may include but are not limited to simulators, code analyzers, test case generators, and path analyzers, and may also include elements used in the software engineering environment.	(ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Software Thread	See Thread.	
Software Under Test	The portion of software that is to be exercised by test verification activities. This could represent an entire CSCI, or only a piece of it.	R. Fulford

Term	Definition	Source
Source Code	(1) Text written in a computer programming language. Such a language is specially designed to facilitate the work of computer programmers, who specify the actions to be performed by a computer mostly by writing source code, which can then be automatically translated to binary machine code that the computer can directly read and execute. (2) Computer instructions and data definitions expressed in a form suitable for input to an assembler, compiler, or other translator.	(1) Wikipedia (2) ISO/IEC 24765:2009 Systems and software engineering vocabulary
Specialty Engineering	Domains that are not typical of the main engineering effort. Hardware engineering and software engineering may be used as major elements in a majority of systems engineering efforts and therefore are not viewed as “special”. Engineering domains such as electromagnetic interference, electrical grounding, safety, security, electrical power filtering/uninterruptible supply, reliability/maintainability/availability, human systems integration, and environmental engineering may be included in systems engineering efforts where they have been identified to address special system implementations. They are then viewed as “specialty engineering.”	adapted from Wikipedia
Specification	(1) A detailed formulation, in document form, which provides a definitive description of a system for the purpose of developing or validating the system. (2) A document that fully describes a design element or its interfaces in terms of requirements (functional, performance, constraints, and design characteristics) and the qualification conditions and procedures for each requirement. (3) A document that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system, component, product, result, or service and, often, the procedures for determining whether these provisions have been satisfied. Examples are: requirement specification, design specification, product specification, and test specification.	(1) ISO/IEC 2382-20:1990 Information technology—Vocabulary—Part 20: System development, 20.01.03 (2) IEEE 1220-2005 IEEE Standard for the Application and Management of the Systems Engineering Process, 3.1.28 (3) A Guide to the Project Management Body of Knowledge (PMBOK® Guide)—Fourth Edition
Specification Tree	(1) A diagram that depicts all of the specifications for a given system and shows their relationships to one another. (2) A hierarchy of specification elements and their interface specifications that identify the elements and the specifications related to design elements of the system configuration that are to be controlled.	(1) ISO/IEC 24765:2009 Systems and software engineering vocabulary (2) IEEE 1220-2005 IEEE Standard for the Application and Management of the Systems Engineering Process, 3.1.29

Term	Definition	Source
Stakeholder	(1) Individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations. (2) Individual, group or organization that can affect, be affected by, or perceive itself to be affected by, a risk. (3) Individual, group, or organization who may affect, be affected by, or perceive itself to be affected by a decision or activity. (4) Person or organization (e.g., customer, sponsor, performing organization, or the public) that is actively involved in the project, or whose interests may be positively or negatively affected by execution or completion of the project. A stakeholder may also exert influence over the project and its deliverables. Note: The decision-maker is also a stakeholder.	(1) ISO/IEC 12207:2008 Systems and software engineering--Software life cycle processes, 4.45; ISO/IEC 15288:2008 Systems and software engineering--System life cycle processes, 4.29; ISO/IEC 15939:2007 Systems and software engineering--Measurement process, 3.37 (2) ISO/IEC 16085:2006 Systems and software engineering--Life cycle processes--Risk management, 3.19 (3) ISO/IEC 38500:2008 Corporate governance of information technology, 1.6.16 (4) A Guide to the Project Management Body of Knowledge (PMBOK® Guide) -- Fourth Edition
Standards	As used in this document, commercial, government, or organizational standard (e.g., IEEE, ISO, company policies, military standards). Distinguish from the term standards and practices.	G. Whittaker
Standards and Practices	Detailed work instructions that the development team follows in conducting their daily activities (e.g., programmer's style guide, check-in/check-out procedures, architectural design patterns). Distinguish from the term standard.	G. Whittaker
Static Analysis	The process of evaluating a system or component based on its form, structure, content, or documentation.	ISO/IEC 24765:2009 Systems and software engineering vocabulary
Stress Scenarios	Tests with the objective to verify the robustness of the software by testing beyond the limits of normal operation. Such tests commonly put a greater emphasis on availability and error handling under a heavy load.	Wikipedia

Term	Definition	Source
System	A composite, at any level of complexity, of personnel, procedures, materials, tools, equipment, facilities and software. The elements of this composite entity are used together in the intended operation or support environment, to perform a given task or achieve a specific production, support or mission requirement.	MIL-STD-882
System Architecture Views	See Architecture Viewpoints.	
System Scenarios	Narrative describing foreseeable interactions of types of users (characters) and the system. Scenarios include information about goals, expectations, motivations, actions and reactions. Scenarios are neither predictions nor forecasts, but rather attempts to reflect on or portray the way in which a system is used in the context of daily activity. Scenarios are frequently used as part of the systems development process. Scenarios are written in plain language, with minimal technical details, so that stakeholders (designers, usability specialists, programmers, engineers, managers, marketing specialists, etc.) can have a common example which can focus their discussions. Increasingly, scenarios are used directly to define the wanted behavior of software: replacing or supplementing traditional functional requirements.	Adapted from Wikipedia
System Testing	System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic. As a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.	Wikipedia
Systems/Software Trade Studies	See Trade Study.	G. Whittaker

Term	Definition	Source
Technical Performance Measure (TPM)	(1) A key indicator of progress, parameter, or a metric that can be used to monitor the progress or performance of selected requirements. A technical performance measure is monitored to ensure that it remains within tolerances as an indication of the progress of the design. (2) 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 measureable target and threshold values. Examples of technical parameters, which can be tracked using TPMs are space vehicle weight, space vehicle power, and computer system resource margins (e.g., CPU, I/O, memory margins).	(1) http://www.argospress.com/Resources/systems-engineering/technizperfomeasu.htm (2) Software Development Standard for Space Systems TOR-2004(3909)-3537 Rev. B, p12.
Technical Software Risk Database Items	Database records containing descriptive details of technical software risks.	G. Whittaker
Test Anomalies	(1) A test condition that deviates from expectations, based on requirements specifications, design documents, user documents, or standards, or from someone's perceptions or experiences. (2) Anything observed in the test operation of software or system that deviates from expectations based on previously verified software products, reference documents, or other sources of indicative behavior.	(1) Adapted from (IEEE 1028-2008 IEEE Standard for Software Reviews and Audits, 3.1). (2) Adapted from (IEEE 829-2008 IEEE Standard for Software and System Test Documentation, 3.1.4).
Test Case	(1) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path, software thread or scenario or to verify compliance with a specific requirement. (2) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item.	(1 - 2) Adapted from IEEE 1012-2004 IEEE Standard for Software Verification and Validation, 3.1.31.
Test Documentation	Documentation of the test plans, RVTM, test procedures, test cases, scenarios, and witnessed test results.	G. Whittaker
Test Environment	Hardware, software, and data configuration necessary to conduct the test case.	Adapted from ISO/IEC 25051:2006 Software engineering -- Software product Quality Requirements and Evaluation (SQuaRE)—Requirements for quality of Commercial Off-The-Shelf (COTS) software product and instructions for testing, 4.8

Term	Definition	Source
Test Like You Fly	(1) TLYF is a pre-launch verification and validation approach that examines all applicable mission and flight characteristics within the intended operational environment and determines the fullest practical extent to which those characteristics can be applied in testing. The application of this philosophy is intended to avoid experiencing anomalous behavior for the first time on orbit and to validate end-to-end operability and performance of the item under test. (2) "Test Like You Fly" is a term that has progressed from being an undefined notion to an assessment and implementation process. Although "test" is included in its title, it is more than just a test approach—it is an acquisition and systems engineering process and a mission assurance and validation tool.	(1) Adapted from 'Space Vehicle Checklist For Assuring Adherence To "Test-Like-You-Fly" Principles', TOR-2009(8591)-15, 30 June 2009, Frank L. Knight. (2) GSAW 2010 Tutorial A: Test Like You Fly (TLYF), Julia White, Lindsay Tilney; The Aerospace Corporation
Test Plan	(1) A plan describing the scope, approach, resources, and schedule of intended test activities. (2) A plan that describes the technical and management approach to be followed for testing a system or component. (3) A plan that establishes detailed requirements, criteria, general methodology, responsibilities, and general planning for test and evaluation of a system. Note: It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning. Typical contents identify the items to be tested, tasks to be performed, responsibilities, schedules, and required resources for the testing activity.	(1 - 2) Adapted from IEEE 1012-2004 IEEE Standard for Software Verification and Validation, 3.1.33 (3) ISO/IEC 2382-20:1990 Information technology--Vocabulary--Part 20: System development, 20.06.09
Test Procedures	(1) Detailed instructions for the setup, execution, and evaluation of results for a given test case. These instructions may be automated. (2) A document containing a set of associated instructions as in (1).	(1 -2) IEEE 1012-2004 IEEE Standard for Software Verification and Validation, 3.1.34
Testability	(1) Extent to which an objective and feasible test can be designed to determine whether a requirement is met. (2) The degree to which a requirement is stated in terms that permit establishment of test criteria and performance of tests to determine whether those criteria have been met. (3) The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.	(1) ISO/IEC 12207:2008 Systems and software engineering--Software life cycle processes, 4.52 (2) IEEE 1233-1998 (R2002) IEEE Guide for Developing System Requirements Specifications, 3.18 (3) ISO/IEC 24765:2009 Systems and software engineering vocabulary

Term	Definition	Source
Thread	As used in this document, identifiable operationally useful function that is individually testable. Each thread is associated with specific hardware or software components that implement its function and is defined by a stimulus and response. Software threads are derived from system threads, scenarios, or communication patterns.	Adapted from Michael S. Deutch, Software Verification and Validation, Prentiss-Hall, 1982, pg 321
Traceability	(1) Degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another. (2) The identification and documentation of derivation paths (upward) and allocation or flow down paths (downward) of work products in the work product hierarchy. (3) Discernable association among two or more logical entities, such as requirements, system elements, verifications, or tasks. Example: the degree to which the requirements and design of a given system element match. (4) Bidirectional association among two or more logical entities that is discernable in either direction (to and from an entity).	(1) (IEEE 1233-1998 (R2002) IEEE Guide for Developing System Requirements Specifications, 3.19) (2) (IEEE 1362-1998 (R2007) IEEE Guide for Information Technology-System Definition -Concept of Operation Document, 3.24) (3–4) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Trade Study	Evaluation of alternatives, based on criteria and systematic analysis, to select the best alternative for attaining determined objectives.	(ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Unit Test	(1) Testing of individual routines and modules by the developer or an independent tester. (2) A test of individual programs or modules in order to ensure that there are no analysis or programming errors. (3) Test of individual hardware or software units or groups of related units.	(1) (ISO/IEC 24765:2009 Systems and software engineering vocabulary) (2) (ISO/IEC 2382-20:1990 Information technology—Vocabulary—Part 20: System development, 20.05.05) (3) (ISO/IEC 24765:2009 Systems and software engineering vocabulary)
Use Case	A use case in software engineering and systems engineering is a description of a system's behavior as it responds to a request that originates from outside of that system. In other words, a use case describes "who" can do "what" with the system in question. The use case technique is used to capture a system's behavioral requirements by detailing scenario-driven threads through the functional requirements.	https://www.thedacs.com/databases/url/key/5086/5164

Term	Definition	Source
User	(1) The ultimate operator of a piece of software or a system. (2) For the purposes of this guide, users include operators of the system in addition to the end users of the data produced by the system.	(1) Adapted from Wikipedia (2) Software Development Standard for Space Systems TOR-2004(3909)-3537 Rev. B, p12.
Validation	(1) The process of providing evidence that the software and its associated products satisfy system requirements allocated to software at the end of each life cycle activity, solve the right problem, and satisfy intended use and user needs. (2) The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers.	(1) (IEEE 1012-2004 IEEE Standard for Software Verification and Validation, 3.1.35) (2) (A Guide to the Project Management Body of Knowledge (PMBOK® Guide)—Fourth Edition)
Verification	(1) The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. (2) Process of providing objective evidence that the software and its associated products comply with requirements (e.g., for correctness, completeness, consistency, and accuracy) for all life cycle activities during each life cycle process (acquisition, supply, development, operation, and maintenance), satisfy standards, practices, and conventions during life cycle processes, and successfully complete each life cycle activity and satisfy all the criteria for initiating succeeding life cycle activities (e.g., building the software correctly).	(1) (A Guide to the Project Management Body of Knowledge (PMBOK® Guide)—Fourth Edition) (2) (IEEE 829-2008 IEEE Standard for Software and System Test Documentation, 3.1.54)
Verification Databases	Databases containing test data used for verification of requirements.	G. Whittaker
Verification Environments	The hardware and software hosting and supporting the verification of software products. Usually intended to simulate or emulate the target environment in various degrees of operational stress, the verification environment also includes extended simulation tools, drivers, and data as necessary. See also: Software Test Environment, Test Environment.	G. Whittaker
Verification Method	Standard methods for verification are inspection, analysis, demonstration, and test.	Software Development Standard for Space Systems TOR-2004(3909)-3537 Rev. B, p12.

Term	Definition	Source
Verification Plans	The methods used for verification, such as analysis, demonstration, inspection, and testing of the products and the processes that produce the products. Usually documented in a test plan.	Adapted from ISO/IEC JTC 1/SC 7 2010-12-22, ISO/IEC FDIS 15289:2010(E) Software and Systems Engineering - Content of life-cycle information products (documentation), item g.
Verification Report	The verification report provides the results and conclusions of verification on a software item, system, or subsystem. It enables the acquirer to assess the verification and its results. It includes system identification and overview, verification requirements and criteria, overview of results, identification of items verified and dates of verification, detailed results, problems encountered, and rationale for decisions.	ISO/IEC/IEEE 15288:2008 reference: 6.4.6.3b and ISO/IEC/IEEE 12207:2008 reference: 7.2.4.3.1.5 as referenced in ISO/IEC FDIS 15289:2010(E) Draft
Waiver	A written authorization to accept a configuration item or other designated item which, during production or after having been submitted for inspection, is found to depart from specified requirements, but is nevertheless considered suitable for use as is or after rework by an approved method.	(ISO/IEC 24765:2009 Systems and software engineering vocabulary)