

RTCA/DO-178B Tables

Appendix B

**Table A-1
Software Planning Process**

Objective		Applicability by SW Level				Output		Control Category by SW level			
Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1 Software development and integral processes activities are defined.	4.1a 4.3	○	○	○	○	Plan for Software Aspects of Certification	11.1	1	1	1	1
		Software Development Plan	11.2	1	1	2	2				
		Software Verification Plan	11.3	1	1	2	2				
		SCM Plan	11.4	1	1	2	2				
		SQA Plan	11.5	1	1	2	2				
2 Transition criteria, inter-relationships and sequencing among processes are defined.	4.1b 4.3	○	○	○							
3 Software life cycle environment is defined.	4.1c	○	○	○							
4 Additional considerations are addressed.	4.1d	○	○	○	○						
5 Software development standards are defined.	4.1e	○	○	○		SW Requirements Standards	11.6	1	1	2	
		SW Design Standards	11.7	1	1	2					
		SW Code Standards	11.8	1	1	2					
6 Software plans comply with this document.	4.1f 4.6	○	○	○		SQA Records	11.19	2	2	2	
		Software Verification Results	11.14	2	2	2					
7 Software plans are coordinated.	4.1g 4.6	○	○	○		SQA Records	11.19	2	2	2	
		Software Verification Results	11.14	2	2	2					

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
Blank		Satisfaction of objective is at applicant's discretion.
	1	Data satisfies the objectives of Control Category 1 (CC1).
	2	Data satisfies the objectives of Control Category 2 (CC2).

Table A-2
Software Development Processes

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	High-level requirements are developed.	5.1.1a	○	○	○	○	Software Requirements Data	11.9	1	1	1	1
2	Derived high-level requirements are defined.	5.1.1b	○	○	○	○	Software Requirements Data	11.9	1	1	1	1
3	Software architecture is developed.	5.2.1a	○	○	○	○	Design Description	11.10	1	1	2	2
4	Low-level requirements are developed.	5.2.1a	○	○	○	○	Design Description	11.10	1	1	2	2
5	Derived low-level requirements are defined.	5.2.1b	○	○	○	○	Design Description	11.10	1	1	2	2
6	Source Code is developed.	5.3.1a	○	○	○	○	Source Code	11.11	1	1	1	1
7	Executable Object Code is produced and integrated in the target computer.	5.4.1a	○	○	○	○	Executable Object Code	11.12	1	1	1	1

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	1	Data satisfies the objectives of Control Category 1 (CC1).
	2	Data satisfies the objectives of Control Category 2 (CC2).

Table A-3
Verification Of Outputs of Software Requirements Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Software high-level requirements comply with system requirements.	6.3.1a	●	●	○	○	Software Verification Results	11.14	2	2	2	2
2	High-level requirements are accurate and consistent.	6.3.1b	●	●	○	○	Software Verification Results	11.14	2	2	2	2
3	High-level requirements are compatible with target computer.	6.3.1c	○	○			Software Verification Results	11.14	2	2		
4	High-level requirements are verifiable.	6.3.1d	○	○	○		Software Verification Results	11.14	2	2	2	
5	High-level requirements conform to standards.	6.3.1e	○	○	○		Software Verification Results	11.14	2	2	2	
6	High-level requirements are traceable to system requirements.	6.3.1f	○	○	○	○	Software Verification Results	11.14	2	2	2	2
7	Algorithms are accurate.	6.3.1g	●	●	○		Software Verification Results	11.14	2	2	2	

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	1	Data satisfies the objectives of Control Category 1 (CC1).
	2	Data satisfies the objectives of Control Category 2 (CC2).

Table A-4
Verification Of Outputs of Software Design Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Low-level requirements comply with high-level requirements.	6.3.2a	●	●	○		Software Verification Results	11.14	2	2	2	
2	Low-level requirements are accurate and consistent.	6.3.2b	●	●	○		Software Verification Results	11.14	2	2	2	
3	Low-level requirements are compatible with target computer.	6.3.2c	○	○			Software Verification Results	11.14	2	2		
4	Low-level requirements are verifiable.	6.3.2d	○	○			Software Verification Results	11.14	2	2		
5	Low-level requirements conform to standards.	6.3.2e	○	○	○		Software Verification Results	11.14	2	2	2	
6	Low-level requirements are traceable to high-level requirements.	6.3.2f	○	○	○		Software Verification Results	11.14	2	2	2	
7	Algorithms are accurate.	6.3.2g	●	●	○		Software Verification Results	11.14	2	2	2	
8	Software architecture is compatible with high-level requirements.	6.3.3a	●	○	○		Software Verification Results	11.14	2	2	2	
9	Software architecture is consistent.	6.3.2b	●	○	○		Software Verification Results	11.14	2	2	2	
10	Software architecture is compatible with target computer.	6.3.3c	○	○			Software Verification Results	11.14	2	2		
11	Software architecture is verifiable.	6.3.3d	○	○			Software Verification Results	11.14	2	2		
12	Software architecture conforms to standards.	6.3.3e	○	○	○		Software Verification Results	11.14	2	2	2	
13	Software partitioning integrity is confirmed.	6.3.3f	●	○	○	○	Software Verification Results	11.14	2	2	2	2

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	1	Data satisfies the objectives of Control Category 1 (CC1).
	2	Data satisfies the objectives of Control Category 2 (CC2).

Table A-5
Verification Of Outputs of Software Coding & Integration Processes

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Source Code complies with low-level requirements.	6.3.4a	●	●	○		Software Verification Results	11.14	2	2	2	
2	Source Code complies with software architecture.	6.3.4b	●	○	○		Software Verification Results	11.14	2	2	2	
3	Source Code is verifiable.	6.3.4c	○	○			Software Verification Results	11.14	2	2		
4	Source Code conforms to standards.	6.3.4d	○	○	○		Software Verification Results	11.14	2	2	2	
5	Source Code is traceable to low-level requirements.	6.3.4e	○	○	○		Software Verification Results	11.14	2	2	2	
6	Source Code is accurate and consistent.	6.3.4f	●	○	○		Software Verification Results	11.14	2	2	2	
7	Output of software integration process is complete and correct.	6.3.5	○	○	○		Software Verification Results	11.14	2	2	2	

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	1	Data satisfies the objectives of Control Category 1 (CC1).
	2	Data satisfies the objectives of Control Category 2 (CC2).

**Table A-6
Testing Of Outputs of Integration Process**

Objective		Applicability by SW Level				Output		Control Category by SW level			
Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1 Executable Object Code complies with high-level requirements.	6.4.2.1	○	○	○	○	Software Verification Cases and Procedures	11.13	1	1	2	2
	6.4.3					Software Verification Results	11.14	2	2	2	2
2 Executable Object Code is robust with high-level requirements.	6.4.2.2	○	○	○	○	Software Verification Cases and Procedures	11.13	1	1	2	2
	6.4.3					Software Verification Results	11.14	2	2	2	2
3 Executable Object Code complies with low-level requirements.	6.4.2.1	●	●	○		Software Verification Cases and Procedures	11.13	1	1	2	
	6.4.3					Software Verification Results	11.14	2	2	2	
4 Executable Object Code is robust with low-level requirements.	6.4.2.2	●	○	○		Software Verification Cases and Procedures	11.13	1	1	2	
	6.4.3					Software Verification Results	11.14	2	2	2	
5 Executable Object Code is compatible with target computer.	6.4.3a	○	○	○	○	Software Verification Cases and Procedures	11.13	1	1	2	2
						Software Verification Results	11.14	2	2	2	2

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	1	Data satisfies the objectives of Control Category 1 (CC1).
	2	Data satisfies the objectives of Control Category 2 (CC2).

Table A-7
Verification Of Verification Process Results

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Test procedures are correct.	6.3.6b	●	○	○		Software Verification Cases and Procedures	11.13	2	2	2	
2	Test results are correct and discrepancies explained.	6.3.6c	●	○	○		Software Verification Results	11.14	2	2	2	
3	Test coverage of high-level requirements is achieved.	6.4.4.1	●	○	○	○	Software Verification Results	11.14	2	2	2	2
4	Test coverage of low-level requirements is achieved.	6.4.4.1	●	○	○		Software Verification Results	11.14	2	2	2	
5	Test coverage of software structure (modified condition/decision) is achieved.	6.4.4.2	●				Software Verification Results	11.14	2			
6	Test coverage of software structure (decision coverage) is achieved.	6.4.4.2a 6.4.4.2b	●	●			Software Verification Results	11.14	2	2		
7	Test coverage of software structure (statement coverage) is achieved.	6.4.4.2a 6.4.4.2b	●	●	○		Software Verification Results	11.14	2	2	2	
8	Test coverage of software structure (data coupling and control coupling) is achieved.	6.4.4.2c	●	●	○		Software Verification Results	11.14	2	2	2	

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	1	Data satisfies the objectives of Control Category 1 (CC1).
	2	Data satisfies the objectives of Control Category 2 (CC2).

Table A-8
Software Configuration Management Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Configuration items are identified.	7.2.1	○	○	○	○	SCM Records	11.18	2	2	2	2
2	Baselines and traceability are established.	7.2.2	○	○	○	○	Software Configuration Index	11.16	1	1	1	1
							SCM Records	11.18	2	2	2	2
3	Problem reporting, change control, change review, and configuration status accounting are established.	7.2.3	○	○	○	○	Problem Reports	11.17	2	2	2	2
		7.2.4					SCM Records	11.18	2	2	2	2
		7.2.5										
		7.2.6										
4	Archive, retrieval, and release are established.	7.2.7	○	○	○	○	SCM Records	11.18	2	2	2	2
5	Software load control is established.	7.2.8	○	○	○	○	SCM Records	11.18	2	2	2	2
6	Software life cycle environment control is established.	7.2.9	○	○	○	○	Software Life Cycle Environment Configuration Index	11.15	1	1	1	2
							SCM Records	11.18	2	2	2	2

- Note: (1) *Although the software configuration management objectives of section 7 do not vary with software level, the control category assigned to the software life cycle data may vary.*
- (2) *The objectives of section 7 provide a sufficient integrity basis in the SCM process activities without the need for the independence criteria.*

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	1	Data satisfies the objectives of Control Category 1 (CC1).
	2	Data satisfies the objectives of Control Category 2 (CC2).

Table A-9
Software Quality Assurance Process

Objective		Applicability by SW Level				Output		Control Category by SW level				
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Assurance is obtained that software development and integral processes comply with approved software plans and standards.	8.1a	●	●	●	●	Software Quality Assurance (SQA) Records	11.19	2	2	2	2
2	Assurance is obtained that transition criteria for the software life cycle processes are satisfied.	8.1b	●	●			SQA Records	11.19	2	2		
3	Software conformity review is conducted.	8.1c 8.3	●	●	●	●	SQA Records	11.19	2	2	2	2

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	1	Data satisfies the objectives of Control Category 1 (CC1).
	2	Data satisfies the objectives of Control Category 2 (CC2).

Table A-10
Certification Liaison Process

Objective		Applicability by SW Level				Output		Control Category by SW level				
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Communication and understanding between the applicant and the certification authority is established.	9.0	○	○	○	○	Plan for Software Aspects of Certification	11.1	1	1	1	1
2	The means of compliance is proposed and agreement with the Plan for Software Aspects of Certification is obtained.	9.1	○	○	○	○	Plan for Software Aspects of Certification	11.1	1	1	1	1
3	Compliance substantiation is provided.	9.2	○	○	○	○	Software Accomplishment Summary	11.20	1	1	1	1
							Software Configuration Index	11.16	1	1	1	1

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	1	Data satisfies the objectives of Control Category 1 (CC1).
	2	Data satisfies the objectives of Control Category 2 (CC2).

Software Roles and Responsibilities

Appendix C

Typical Roles and Responsibilities of the FAA Software Team

(Version 2 – 1/22/01)

Table of Contents

1.0	INTRODUCTION	3
2.0	ROLES AND RESPONSIBILITIES FOR THE SW-ASE.....	3
2.1	ROLES AND RESPONSIBILITIES OF SW-ASE'S IN SOFTWARE APPROVALS UNDER THE TC/ATC/STC PROCESSES	3
2.1.1	<i>Communication and Planning</i>	3
2.2.2	<i>Implementation</i>	4
2.2.3	<i>Future Planning and Involvement</i>	5
2.3	ROLES AND RESPONSIBILITIES OF SW-ASE'S IN THE SOFTWARE ASPECTS OF THE PRODUCTION CERTIFICATE PROCESS.	5
2.4	ROLES AND RESPONSIBILITIES OF SW-ASE'S IN THE SOFTWARE ASPECTS OF THE PARTS MANUFACTURER APPROVAL (PMA) PROCESS.....	5
2.5	ROLES AND RESPONSIBILITIES OF SW-ASE'S IN THE SOFTWARE ASPECTS OF THE TECHNICAL STANDARD ORDER AUTHORIZATION (TSOA) PROCESS.....	6
2.5.1	<i>Description of the TSOA Process</i>	6
2.5.2	<i>Evaluating Capability</i>	7
2.5.3	<i>Issuance of a TSOA</i>	7
2.5.4	<i>Roles and Responsibilities in the TSOA Process</i>	8
2.6	ROLES AND RESPONSIBILITIES OF SW-ASE'S IN THE SOFTWARE ASPECTS OF THE ACSEP PROCESS	8
2.7	ROLES AND RESPONSIBILITIES FOR SW-ASE'S IN THE SOFTWARE ASPECTS OF THE CERTIFICATE MANAGEMENT PROCESS	9
2.8	ROLES AND RESPONSIBILITIES FOR SW-ASE'S IN THE SOFTWARE ASPECTS OF THE DESIGNEE MANAGEMENT PROCESS	10
3.0	ROLES AND RESPONSIBILITIES FOR SOFTWARE NRS.....	11
3.1	NRS TECHNICAL LEADER ROLES AND RESPONSIBILITIES:.....	11
3.2	NRS CERTIFICATION SOFTWARE TEAM ADVISOR ROLES AND RESPONSIBILITIES:	11
4.0	ROLES AND RESPONSIBILITIES FOR SOFTWARE TS.....	12
4.1	TS TECHNICAL EXPERT ROLES AND RESPONSIBILITIES:	12
4.2	TS CERTIFICATION SOFTWARE TEAM MEMBER ROLES AND RESPONSIBILITIES:.....	12
5.0	ROLES AND RESPONSIBILITIES FOR DIRECTORATE STAFF	13
5.1	THE DIRECTORATE STAFF ASSUMES THESE ROLES AND RESPONSIBILITIES:.....	13
6.0	ROLES AND RESPONSIBILITIES FOR HEADQUARTERS STAFF	14

1.0 Introduction

This document describes the roles and responsibilities of the FAA's software team. The team may include the following members:

- Aviation Safety Engineer responsible for the software approval (SW-ASE),
- National Resource Specialist (NRS),
- Technical Specialist (TS),
- Directorate personnel, and/or
- Headquarters personnel.

The typical roles and responsibilities for each team member will be discussed below.

2.0 Roles and Responsibilities for the SW-ASE

The SW-ASE is the ACO engineer responsible for software review and approval. This section describes the roles and responsibilities for SW-ASE's for each of the following processes:

- (1) Type Certificate, Amended Type Certificate, Supplemental Type Certificate (TC/ATC/STC) Process
- (2) Production Certificate (PC) Process
- (3) Parts Manufacturer Approval (PMA) Process
- (4) Technical Standard Order Authorization (TSOA) Process
- (5) Aircraft Certification Systems Evaluation Program (ACSEP) Process
- (6) Certificate Management Process
- (7) Designee Management Process

2.1 Roles and Responsibilities of SW-ASE's in Software Approvals under the TC/ATC/STC Processes

The process for approving software in TC/ATC/STC projects involves three roles for the SW-ASE: (1) communicating with the applicant and planning the project, (2) implementing the review and approval, and (3) determining the future level of involvement based on lessons learned.

2.1.1 Communication and Planning

At the beginning of a project, the SW-ASE should carry out communication with the applicant in order to plan the workload, number of software reviews, amount of delegation, etc. The roles and responsibilities of the SW-ASE during the communication and planning process are shown in Table 1.

TABLE 1 - Roles and Responsibilities for SW-ASE’s in the Communication and Planning for the Software Aspects of the TC/ATC/STC Process

- Participate in Type Board/Familiarization meeting.
- Determine level of FAA involvement for software aspects of project.
- Assess if unique design or new technology is being proposed (to determine if NRS, TS, Directorate personnel, or HQ personnel should be involved).
- Determine designee utilization and resource availability.
- Coordinate software effort with Project Manager.
- Determine software level(s) based on System Safety Assessment (SSA).
- Determine the software life cycle data to be submitted.
- Review PSAC.
- If necessary, review the Software Configuration Management Plan (SCMP), the Software Quality Assurance Plan (SQAP), the Software Development Plan (SDP), and the Software Verification Plan (SVP).
- Provide comment to applicant and obtain resolution of plan deficiencies.
- Provide software input to CPP or equivalent project level plan, including designee delegation plans and interactions during the project.
-
- Resolve any discrepancies in plans with applicant.

2.2.2 Implementation

Implementation of a project is the process of assuring the applicant’s software life cycle processes comply with their approved plans and approving their data submittals after determining compliance with DO-178B or other acceptable means. Implementation may require on-site software reviews, desk-top reviews, and review of designee findings by the SW-ASE. SW-ASE’s roles and responsibilities for implementation of the software review and approval are shown in Table 2.

TABLE 2 - Roles and Responsibilities of SW-ASE’s in the Implementation of the TC/ATC/STC Process

- Approve PSAC and, if necessary, SCMP, SQAP, SDP, and SVP.
- Monitor the applicant’s compliance to their plans.
- Resolve applicant process discrepancies with the approved software plans and DO-178B or acceptable alternative.
- Coordinate tasks to support desk-top and on-site reviews.
- Perform on-site review, desk-top review, designee delegation, or a combination.
- Coordinate with systems certification Software Team.
- Identify and request specific conformity requirements.
- Approve Software Configuration Index (SCI) and Software Accomplishment Summary (SWAS).
- Identify discrepancies and coordinate resolution.
- Identify process improvement opportunities.

2.2.3 Future Planning and Involvement

At the end of each software review or approval, the SW-ASE may want to identify areas for both the FAA and the applicant to improve upon.

2.3 Roles and Responsibilities of SW-ASE's in the Software Aspects of the Production Certificate Process.

The Production Certificate (PC) process begins with the application. The normal process for issuance of a PC is to follow Order 8120.2A, "Production Approval and Surveillance Procedures." The cognizant MIDO, MISO, or CMO may conduct a preliminary audit of the applicant's Quality Control (QC) system and production facilities to ensure compliance with the applicable Code of Federal Regulations (CFR) and policy. The PC project may be assessed during the preliminary audit to determine whether the applicant is involved in airborne software development and computer aided design, manufacturing, inspection, and test (CADMIT) tools. The FAA assesses the project to ensure that the airborne and CADMIT software is addressed in the QC and SCM systems. A Production Certification Board (PCB) may be convened for initial production approvals to evaluate the preliminary audit findings and recommendations from the cognizant MIDO, MISO, or CMO.

The issuance of the PC is primarily the responsibility of the MIDO, MISO, or CMO. However, the SW-ASE might be requested to assist the manufacturing office in evaluation of automated inspection or test equipment used to verify type design.

2.4 Roles and Responsibilities of SW-ASE's in the Software Aspects of the Parts Manufacturer Approval (PMA) Process.

The PMA process begins with the application for PMA. The normal process for issuance of PMA is to follow Order 8110.42, "Parts Manufacturer Approval Procedures." This process applies to anyone producing replacement or modification parts for sale for installation on type certified products. Applicants may obtain design approval on replacement or modification parts through Identicality, or Licensing Agreements. Production manufacturing approval is obtained through the MIDO or MISO inspector's acceptance of the applicant's fabrication inspection system and evaluation of applicant's facility to determine applicant's compliance to 14 CFR part 21, Subpart K.

Since software has some unique characteristics, notice 8110.79, Guidelines for the approval of Field-Loadable Software by Finding Identicality through the Parts Manufacturer Approval Process, identifies the PMA process for field-loadable software. Field-loadable software is where PMAs are typically desired for software. At present, the test and computation approach is not supported for PMA software.

Typical Roles and Responsibilities of the Software Team (Version 2 – 1/22/01)

Tables 3 describe the roles and responsibilities for the SW-ASE to be performed for: (1) PMA application, (2) approval by identity with licensing agreement, and (2) approval by identity without licensing agreement. This specifically applies to PMA for software. Reference notice 8110.79 as needed.

Function	SW-ASE Roles and Responsibilities
PMA Application	<ul style="list-style-type: none"> • Determine level of FAA software involvement. • Determine designee utilization and resource availability. • Coordinate software effort with Project Manager. • Coordinate with the systems certification Software Team. • Establish certification basis. • Participate in familiarization and technical meetings. • Review applicant’s software plans. • Determine if unique design or new technology warrants coordination with NRS, TS, Directorate, or Headquarters personnel. • Resolve plan discrepancies with the applicant. • Perform on-site reviews, desk-top reviews, designee delegation, or combination, as necessary. • Identify discrepancies. • Review SCI and SWAS.
Identity With Licensing Agreement	<ul style="list-style-type: none"> • SW-ASE is typically not involved, unless requested by the manufacturing office.
Identity Without Licensing Agreement	<ul style="list-style-type: none"> • Specify software life cycle data to be submitted. • Review submitted software life cycle data and resolve discrepancies with applicant. • Verify approved software configuration.

Table 3. Roles and Responsibilities for SW-ASE’s in the PMA Process

2.5 Roles and Responsibilities of SW-ASE’s in the Software Aspects of the Technical Standard Order Authorization (TSOA) Process.

The TSOA is a joint authorization by both the ACO and MIDO or MISO, and has many similarities to the TC/ATC/STC process. The normal process to obtain a TSOA is to follow Order 8150.1A. However, the TSOA process also has some unique characteristics that are described below.

2.5.1 Description of the TSOA Process

The TSOA is an authorization to manufacture equipment that meets TSO-specified requirements; it is not approval to install the equipment on an aircraft or engine. The design portion of the TSOA process is responsibility of the applicant. The applicant submits the TSO data package and a statement of compliance to the ACO. Most TSO

Typical Roles and Responsibilities of the Software Team (Version 2 – 1/22/01)

authorizations are granted based on a review of the data package, reliance on the applicant's statement of compliance, and an evaluation of the capability of the applicant to produce the TSO equipment. FAA acceptance of TSO systems with embedded software is based on a review of the TSO data package for compliance with RTCA DO-178[] or other acceptable means, as well as the applicant's statement of compliance that the TSO article meets the performance specifications of the TSO. Once the TSO is granted, the TC of the aircraft may need to be amended or supplemented to allow the TSOA equipment to be installed on the aircraft. Granting a TSOA is, in and of itself, not sufficient substantiation to amend a TC; installation substantiation is required also.

The TSOA process may begin with an initial familiarization meeting, a letter of intent or application, where the applicant's project schedule and plans are discussed. Applicants should be encouraged to seek software expertise and FAA involvement early in the project. The FAA can provide guidance on software compliance and certification concerns. The applicant may want to discuss with the FAA such areas as: the certification plans; especially the PSAC, the system safety assessment, human factors issues, failure condition categories, software levels, software and hardware partitioning, etc.

The SW-ASE evaluation begins after the submission of the completed TSO data package. TSO requirements sometimes specify the data submittal requirements. If they don't, the applicant should submit the PSAC, SCI, and SWAS. The ACO SW-ASE may request additional data be submitted. The evaluation consists of the following:

1. review of applicant's statement of compliance;
2. review TSO data submittals, including software life cycle data; and
3. recommend approval or denial of deviations.

The manufacturing office will evaluate the QC manual for compliance with the applicable CFR, policy, and verification of implementation compliance with the manual.

2.5.2 Evaluating Capability

As part of assessing the applicant's capability to make statements of compliance, the FAA must assess the company's capability to produce software in compliance with the appropriate software level of DO-178[]. The assessment may be accomplished through an FAA software review conducted by a team. Once the FAA has determined the applicant capable, the applicant may be deemed "capable" for that level of software.

2.5.3 Issuance of a TSOA

If the Software Team finds all submittals from the applicant acceptable, the TSOA is issued. If the applicant's request is denied, the reason for denial should be communicated to the applicant. When acceptable corrections are made, the TSOA may be issued. Deviations are evaluated by engineering and a recommendation to approve or deny, with substantiating data, is provided to Headquarters (AIR-100) for concurrence. AIR-100

Typical Roles and Responsibilities of the Software Team (Version 2 – 1/22/01)

will communicate approval or denial of the deviations to the local ACO who provides a formal response to the applicant.

2.5.4 Roles and Responsibilities in the TSOA Process

Table 4 below describes roles and responsibilities of the SW-ASE in TSOA project familiarization and evaluation.

TABLE 4 - Roles and Responsibilities for SW-ASE for Software Aspects of the TSOA Project
<ul style="list-style-type: none">• Review applicant statement of compliance and TSO data package submitted with TSO application.• Participate in familiarization meeting•• Determine software level of FAA involvement for TSO.• Review software life cycle data of TSO data package.• Assess software level acceptability.• Request additional software data to be submitted as necessary to substantiate compliance.• Perform on-site or desk reviews, as necessary to substantiate compliance.• Evaluate deviation requests, send recommendations to AIR-100, and forward resolution to applicant.• Resolve any discrepancies with the applicant.• Send TSOA letter to applicant.

2.6 Roles and Responsibilities of SW-ASE's in the Software Aspects of the ACSEP Process

The production approval holder's SQA and SCM processes and Quality Control system are evaluated to the criteria found in Order 8100.7, "Aircraft Certification Systems Evaluation Program (ACSEP)." Individuals assigned to review the software sub-system might comprise of one or more SW-ASE's and/or Aviation Safety Inspectors (ASI), possibly flight test pilots. If more than one individual is participating in the review, than one will be assigned the role of software team leader. Table 5 defines the roles and responsibilities for the SW-ASE or ASE who is performing the software aspects of ACSEP evaluations.

TABLE 5 - Roles and Responsibilities for Software Aspects of the ACSEP Evaluation
<ul style="list-style-type: none">• Examine the software quality process per the ACSEP order.• Document findings and observations.• Monitor corrective actions.

2.7 Roles and Responsibilities for SW-ASE’s in the Software Aspects of the Certificate Management Process

The Certificate Management process begins with the issuance of a new approval, a scheduled visit, or information from manufacturing. Certificate Management for systems with software (and the scheduled visits portion of certificate management) is an activity for both engineering and manufacturing inspection. Certificate Management is an ongoing process that applies to TC/ATC/STC, PC, TSOA, and PMA products.

Certificate Management of software systems should be proactive and may include:

- evaluation of the software development processes, if not previously reviewed (TC/ATC/STC/TSOA);
- evaluation of the SCM change process (e.g., design change, change control, baseline change, specification change notices, etc.);
- an evaluation of the SCM data retention and retrieval;
- verification that the software can be built, linked, and loaded into production units using approved procedures;
- analysis of product service history, including problem reports, accident/incident databases, Airworthiness Directives databases, System Deficiency Report databases to aid in determining the quality of the original development subsequent changes. This provides feedback to FAA manufacturing and engineering offices for continuous improvement activities;
- an assurance that manufacturing, test, and inspection software is controlled in compliance with the QA system and SCM; and
- Reevaluate SQA and SCM processes to ensure continued acceptability.

The above activities may result in a report of findings relevant to compliance with Order 2150.3A, “Compliance and Enforcement Program”, from the ACO or MIDO.

Table 6 defines the typical roles and responsibilities for the SW-ASE for the software aspects of Certificate Management.

TABLE 6 – SW-ASE Roles and Responsibilities Certificate Management Process

- | |
|---|
| <ul style="list-style-type: none">• Review Service Difficulty Reports for software related trends.• Approve Service Bulletins.• Draft Airworthiness Directives.• Discuss SQA and SCM deficiencies with applicant.• Evaluate the software life cycle processes, if problems arise. |
|---|

2.8 Roles and Responsibilities for SW-ASE's in the Software Aspects of the Designee Management Process

Much of the software aspects of certification are delegated to the Designated Engineering Representative (DER). The process of managing designees who perform software functions needs to take into consideration the following:

- (1) Designee qualification, selection, and orientation.
- (2) Oversight of designee usage on projects.
- (3) Oversight of designee approval and activities.
- (4) Designee renewal and evaluation.
- (5) Training of designees.

Table 7 defines the roles and responsibilities for the Designee advisor and SW-ASE to be performed for the software aspects of Designee Management.

TABLE 7 - Roles and Responsibilities of SW-ASE for Designee Management
<ul style="list-style-type: none">• Evaluate designee qualifications to the criteria of the appropriate Order.• Participate in training and mentoring activities to prepare the designees.• Apply the designee appointment and renewal procedures required by FAA Orders.• Evaluate level of designee activity.

3.0 ROLES AND RESPONSIBILITIES FOR SOFTWARE NRS

The NRS provides professional technical guidance, advice and assistance within the FAA and to the aviation industry. They are the FAA's direct link to an extensive professional network in the research and development community, professional and academic organizations, private industry, other government and regulatory authorities, and national and international experts in the field of software. The NRS operates in the role of technical leader and certification Software Team advisor. The roles and responsibilities of the NRS in both capacities are described below:

3.1 NRS Technical Leader Roles and Responsibilities:

- Consults on programs that are applying new technology.
- Initiates and serves on committees regarding standardization of new technology areas.
- Addresses issues that require precedent setting approaches to policy and means of compliance.
- Assists Directorate and Headquarters staffs in understanding technology and related issues in order to develop rules and policy guidance.
- Educates Headquarters, Technical Specialists, Directorate Staff, SW-ASE's, SW-ASI's and Designees regarding new technology compliance issues.
- Conducts research and development in the areas of specialty and responsibilities.

3.2 NRS Certification Software Team Advisor Roles and Responsibilities:

- Attends familiarization meetings, when requested.
- Advise the Software Team on issues that require precedent setting approaches to policy and means of compliance.
- Participates in Special Certification Reviews, Critical Design Reviews, and Multiple Expert Opinion Software Teams.
- Participates in formal technical Software Team meetings.
- Provides timely response to Software Team for methods of compliance or precedent-setting design features.
- Assists SW-ASE, SW-ASI, and applicant in understanding new technology and related issues and identifying means of compliance.

4.0 ROLES AND RESPONSIBILITIES FOR SOFTWARE TS

The TS provides technical expertise to the FAA in the area of software and acts as the focal point for issues of software technology. The TS is responsible for being current on the latest technologies, methods, and policies by working closely with the NRS, Standards and MIO staff of the Directorates, and Headquarters. The TS operates in the role of technical expert and certification Software Team member. The roles and responsibilities of the TS in each of these capacities are described below:

4.1 TS Technical Expert Roles and Responsibilities:

- Assists ACO's, MIDO's, Directorate Staff, and Headquarters in establishing policy and procedures regarding software issues.
- Participates in meetings with the NRS and industry.
- Mentoring and assists SW-ASE and SW-ASI on software issues.
- Provides an evaluation of the SQA subsystem when requested to participate on an ACSEP review.
- Participates on industry Software Teams to establish standards and guidance.
- Provides expertise within discipline.

4.2 TS Certification Software Team Member Roles and Responsibilities:

- Participates in projects involving new technology or new application of technology.
- When requested by ACO, evaluates software life cycle processes during certification projects.
- When requested by ACO, evaluates SCM and SQA processes for airborne systems and manufacturing operations to assure post-certification compliance.
- Identifies compliance issues.
- When requested by ACO, conducts software reviews and inspections.
- Provides technical recommendations to SW-ASE's and SW-ASI's.

5.0 ROLES AND RESPONSIBILITIES FOR DIRECTORATE STAFF

The Directorate Staff consists of both the standards staff and the manufacturing inspection office. The Directorate Staff provides part-specific and project-specific rules and policy to the certification Software Team. They are also the focal point within the Directorate for policy.

5.1 The Directorate Staff assumes these roles and responsibilities:

- Provides input to Headquarters to ensure national policy is consistent with Directorate (Part 23, 25, 27, 29, 33, 35) policy.
- Participate in familiarization meetings for significant projects.
- Identifies and clarifies software policy for the ACO's and MIDO's.
- Assists the ACO's and MIDO's in formalizing their concerns with policy implementation problems to Headquarters.
- Encourages and ensures standardized application of national policy and regulations.
- Encourages the definition of design features and methods of compliance early in the project.
- Represents the Directorate at technical forums and meetings that involve software.
- Assists Headquarters in the development of regulations and national policy.
- Recommends issues requiring national policy to Headquarters.
- Participates in software reviews, as requested.
- Provides software process evaluation expertise as project Software Team member.
- Works with the NRS and TS on national software issues.
- Serves as technical expert, as requested.

6.0 ROLES AND RESPONSIBILITIES FOR HEADQUARTERS STAFF.

The Headquarters staff assumes the following roles and responsibilities for software aspects of certification:

- Serves as focal point working with Directorate Staff, NRS, TS, ACO's and MIDO's to ensure policy and guidance standardization among all Directorates.
- Develops new policy, guidance, and regulations based on input from NRS, TS, Directorate Staff, ACO's, MIDO's, and Industry.
- Interprets and explains policy and guidance to the Directorate Staff, ACO's and MIDO's
- Serves as liaison among different FAA communities.
- Participates in projects that require changes or additions to national software policy.
- Develops national training programs to promote standardization throughout AIR.
- Sponsors national software standardization conferences.
- Manages Research and Development programs involving software.
- Promotes international harmonization.
- Serves as the federal representative on national software committees.
- Works closely with Headquarters management Software Teams.
- Serves as technical expert during a software review, as requested.

**Excerpts from FAA Notice 8110.87
(Level of FAA Involvement in Software
Projects)**

Appendix D

DO-178B Software Level	Level of FAA Involvement
D	LOW
C	LOW or MEDIUM
B	MEDIUM or HIGH
A	MEDIUM or HIGH

Table 1. Software Level Criteria

	CRITERIA	Scale	MIN.	MAX.	Score
1.	Applicant/Developer Software Certification Experience				
1.1	Experience with civil aircraft and systems certification.	Scale:	0	5	10
		# projects:	0	3-5	6+
1.2	Experience with DO-178B.	Scale:	0	5	10
		# projects:	0	2-4	5+
1.3	Experience with DO-178 or DO-178A.	Scale:	0	3	5
		# projects:	0	4-6	7+
1.4	Experience with other software standards (other than DO-178 [])	Scale:	0	2	4
		# projects:	0	4-6	7+
2.	Applicant/Developer Demonstrated Software Development Capability				
2.1	Ability to consistently produce DO-178B software products.	Scale:	0	5	10
		Ability:	Low	Med	High
2.2	Cooperation, openness and resource commitments	Scale:	0	5	10
		Ability:	Low	Med	High
2.3	Ability to manage software development and sub-contractors	Scale:	0	5	10
		Ability:	Low	Med	High
2.4	Capability assessments (e.g., SEI CMM, ISO 9001-3, IEC)	Scale:	0	2	4
		Ability:	Low	Med	High
2.5	Development team average relevant experience	Scale:	0	5	10
		Ability:	< 2 yrs	2-4 yrs	> 4 yrs
3.	Applicant/Developer Software Service History				
3.1	Incidents of software-related problems. (as a percentage of affected products)	Scale:	0	5	10
		Incidents:	> 25%	> 10%	None
3.2	Company management and support of designees	Scale:	0	5	10
		Quality:	Low	Med	High
3.3	Company software quality assurance organization and configuration management process	Scale:	0	5	10
		Quality:	Low	Med	High
3.4	Company stability and commitment	Scale:	0	3	6
		Stability:	Low	Med	High
3.5	Success of past company certification efforts	Scale:	0	3	6
		Success:	None	>50%	All
4.	The Current System and Software Application				
4.1	Complexity of the system architecture, functions and interfaces	Scale:	0	5	10
		Complex:	High	Med	Low
4.2	Complexity & size of the software and safety features	Scale:	0	5	10
		Complex:	High	Med	Low
4.3	Novelty of design and use of new technology	Scale:	0	5	10
		Newness:	Much	Some	None
4.4	Software development and verification environment	Scale:	0	3	6
		Environ:	None	Older	Modern
4.5	Use of alternative methods or additional considerations	Scale:	0	3	6
		Standard:	Much	Little	None

Table 2. Other Relevant Criteria

5.	Designee Capabilities		
5.1	Experience of designees with DO-178B.	Scale: 0 5 10 Projects: <5 5-10 >10	
5.2	Designee authority, autonomy and independence.	Scale: 0 5 10 Autonomy: None Self-starter Outgoing	
5.3	Designee cooperation, openness and issue resolution effectiveness.	Scale: 0 5 10 Effectiveness: Responsive Cooperative Outgoing	
5.4	Relatedness of assigned designee's experience.	Scale: 0 5 10 Related: None Somewhat Exact	
5.5	Designees workload on project and other projects.	Scale: 0 5 10 Projects: <5 5-10 >10	
5.6	Experience of designees with other software standards (other than DO-178[]).	Scale: 0 3 5 Projects: <5 5-10 >10	

Total Score Result (TSR): _____

Table 2. Other Relevant Criteria (Continued)

Total Score Result (TSR) (from Table 1)	Software Level A	Software Level B	Software Level C	Software Level D
$TSR \leq 80$	HIGH	HIGH	MEDIUM	LOW
$80 < TSR \leq 130$	HIGH	MEDIUM	MEDIUM	LOW
$130 < TSR$	MEDIUM	MEDIUM	LOW	LOW

Table 3. Level of Involvement Determination

TABLE 4. TYPICAL PROGRAM DECISIONS BASED ON LOFI OUTCOME

Level of FAA Involvement	Typical Program Decisions
HIGH	<ul style="list-style-type: none"> ▪ Minimal delegation to designees (i.e., Designee may recommend approval of some data and approve other type design data). ▪ NRS, Technical Specialist (TS), Directorate staff, and/or Headquarters (HQ) staff involvement is likely. ▪ FAA involvement throughout the software life cycle, including mentoring, on-site reviews and desk reviews (recommend no less than 2 on-site reviews). ▪ Submittal of all plans: Plan for Software Aspects of Certification (PSAC), Software Development Plan (SDP), Software Verification Plan (SVP), Software Configuration Management Plan (SCMP), and Software Quality Assurance Plan (SQAP). ▪ Submittal of Software Accomplishment Summary (SAS), Software Configuration Index (SCI) and Verification Results. ▪ Submittal of DO-178B Objectives Compliance Matrix (reference FAA Job Aid, “Conducting Software Reviews Prior to Certification,” dated June 1998), which may be submitted as part of the SAS.
MEDIUM	<ul style="list-style-type: none"> ▪ Moderate delegation to designees (i.e., Designee may recommend approval of PSAC and SAS; Designee may approve SCI; and Designee may approve SVP, SDP, SQAP, SCMP, and other data). ▪ Involvement at least initially (planning, regulation and policy interpretation, some mentoring) and toward the end of the project (final compliance). ▪ NRS, TS, Directorate staff, or Headquarters staff involvement may be needed. ▪ Conduct at least 1 on-site review but mostly desk reviews of data. ▪ Require submittal of PSAC, SCI, SAS. ▪ May request submittal of SVP, SQAP, SCMP, and SDP.
LOW	<ul style="list-style-type: none"> ▪ Maximum delegation to designees (i.e., Designee may recommend approval of PSAC and designee may approve all other data/documents.) ▪ Minimal FAA involvement (e.g., no on-site reviews, little or no desk reviews). ▪ Rarely need NRS, TS, Directorate staff, or HQ staff involvement. ▪ Submittal of PSAC, SCI, and SAS.

NOTE: Table 4 is only an example of High, Medium, and Low decisions. Each program will have slightly different needs.

APPENDIX 1. LEVEL OF FAA INVOLVEMENT (LOFI) WORKSHEET

Applicant: _____ **Project Name/Number:** _____
ACO Engineer: _____ **System Type:** _____
MIDO/MISO Inspector: _____ **Software Level:** _____
DER Name: _____ **Date of Assessment:** _____
TSR (from Table 2): _____ **Other Info:** _____
Resulting LOFI: _____ **Policy Issues:** _____

Plan Based on LOFI Assessment: (e.g., number of FAA on-site reviews, number of FAA desk reviews, data to be submitted to the FAA, delegation to DERs, etc.)

Mid-Project Corrections: (based on project improvements or problems)

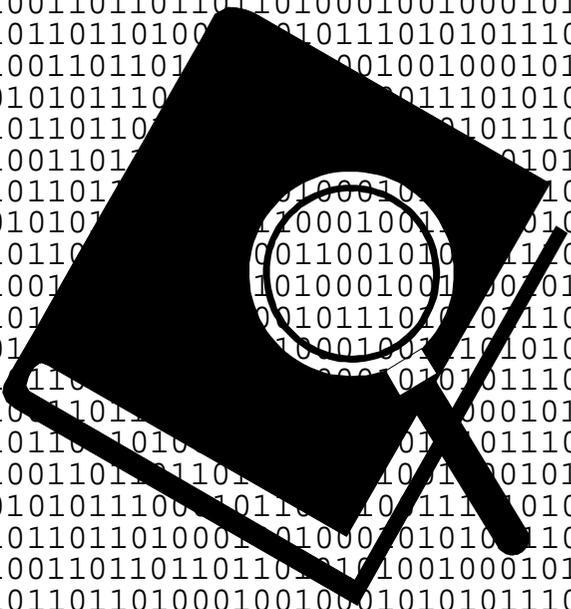
Actual Project Results: (e.g., number of FAA on-site reviews, number of FAA desk reviews, data submitted to the FAA, delegation to DERs, etc.)

Excerpts From Software Review Job Aid

Appendix E



Conducting Software Reviews Prior to Certification



Job Aid

AIRCRAFT CERTIFICATION SERVICE

June 1998

JOB AID – TABLE OF CONTENTS

	Page #
Job Aid Layout	ii
Acronyms	iii
I. Introduction	I-1
Purpose.....	I-1
Organization.....	I-3
Stakeholders in the Software Review Process	I-3
II. Overview of the Software Review Process	II-1
Review Types.....	II-1
Stages of Involvement	II-2
III. Getting Started	III-1
Determining Level of Involvement.....	III-1
Overview of Common Tasks	III-1
Teaming of Engineers and Inspectors	III-2
IV. Tasks Involved in the Software Review	IV-1
Overview.....	IV-1
Task 1: Preparing for the Software Review	IV-2
Task 2: Performing the Software Review and Documenting Findings and Observations	IV-5
Task 3: Preparing and Conducting Exit Briefing.....	IV-8
Task 4: Conducting Follow-Up Activities.....	IV-10
V. Activities for Stages of Involvement	V-1
Planning	V-2
Development Process.....	V-11
Verification/Test	V-23
Final Review	V-30

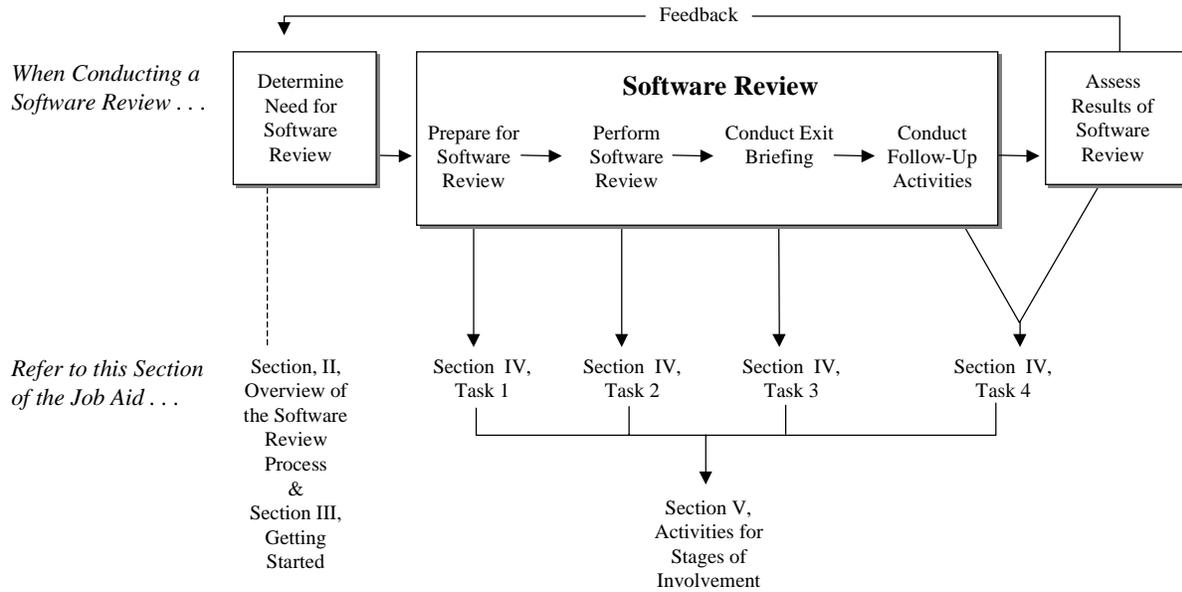
APPENDICES

Appendix A - Sample Notification Letter and Meeting Agendas.....	A-1
Appendix B - Summary of Compliance Findings/Observations.....	B-1
Appendix C - Sample Software Review Transmittal Letter and Report.....	C-1

TABLES

Table 1. Stakeholders in the Software Review Process	I-4
Table 2. On-Site/Desk-Top Review Summary	II-1
Table 3. Delegation of Software Reviews	II-2
Table 4. Summary of DO-178B Tables	II-2
Table 5. Overview of Stages of Involvement.....	II-4

Job Aid Layout



JOB AID
STAGE OF INVOLVEMENT #1 – ACTIVITIES/QUESTIONS

Item #	Evaluation Activity/Question	ASE-SW/ ASI-SW	DO-178B objective	Reviewed? (✓) Issue? (✓*)
1.1	Review all plans (PSAC, SCMP, SQAP, SDP, SVP, software tool plans, etc.). Based on your review of all the plans, consider the following questions:			
1.1.1	Has the planning data been signed and put under CM? Verify there is objective evidence of coordination (e.g., authorized signatures) from all organizations controlled and affected by the software plan.	*ASI-SW/ ASE-SW	A-1, #1-7	
1.1.2	Are plans and standards cited complete, clear, and consistent?	*ASE-SW/ ASI-SW	A-1, #1,7	
1.1.3	Do the plans state procedures for implementing software changes? <ul style="list-style-type: none"> • Are any criteria established for minor/major changes? • If the project is a change to existing software, is it a minor or major change? • If major, has the applicant outlined a procedure for change impact analysis? • Does the SVP address testing in event of major change? • Do company procedures allow for regression testing analysis? 	*ASE-SW/ ASI-SW	A-1, #1,2	
1.1.4	Are the inputs, outputs, and data flows specified for each process?	*ASE-SW/ ASI-SW	A-1, #1	
1.1.5	Are the development and verification life cycle activities defined in sufficient detail (reference DO-178B sections 11.1-11.3) to satisfy section 4.2.	*ASE-SW/ ASI-SW	A-1, #1-7	

Item #	Evaluation Activity/Question	ASE-SW/ ASI-SW	DO-178B objective	Reviewed? (✓) Issue? (✓*)
1.1.6	Do the plans meet the DO-178B planning objectives in Table A-1? (i.e., Is each plan internally consistent? Are the plans consistent with each other? Is the software life cycle defined? Are the transition criteria defined?)	*ASE-SW/ ASI-SW	A-1, #7	
1.1.7	If the plans are followed, would this assure that all DO-178B objectives in Tables A-2 through A-10 are met? (Consider each 178B objective after your comprehensive reading of the plans.)	*ASE-SW/ ASI-SW	A-2 to A-10 (all objectives)	
1.2	Determine if additional considerations defined in Section 12 of DO-178B have been documented and addressed in the plans. Consider the following questions:			
1.2.1	Does the use of tools result in the elimination, reduction, or automation of processes found in DO-178B? Verify that any software tools that are required are identified and that how the tools are to be used is documented.	ASE-SW	A-1, #3	
1.2.2	<p>Are tools supported with a tool qualification plan? Verify that tools are properly categorized into development, configuration management, or verification tools. Verify that the plan for qualification of tools is documented and adequate for the specified tool use.</p> <p><u>Note 1:</u> Development tools can introduce an error and should follow the criteria of DO-178B, Paragraph 12.2.1. This data should be reviewed unless previously qualified, have not undergone change, and are being applied in the same manner.</p> <p><u>Note 2:</u> Verification tools can fail to detect errors and are required to meet the operational/function requirements as described in DO-178B, paragraph 12.2.3.2. This data should be reviewed unless previously qualified, have not undergone change, and are being applied.</p>	ASE-SW	A-1, #3	

Item #	Evaluation Activity/Question	ASE-SW/ ASI-SW	DO-178B objective	Reviewed? (✓) Issue? (✓*)
1.2.3	Are such items as previously developed software, COTS, user-modifiable software, field-loadable software, option-selectable software, multiple-version dissimilar software, product service history, alternative methods of compliance, etc. adequately addressed in the plans?	*ASE-SW/ ASI-SW	A-1, #4 (Section 2.3, 2.4, 2.5, and 12)	
1.2.4	Have any issues regarding modification to legacy systems or reuse of legacy software been addressed in the plans? (Reference notice for use of DO-178B for legacy systems.)	ASE-SW	A-1, #4	
1.2.5	Has a NRS, Directorate, and/or Headquarters personnel reviewed unique additional considerations (if required)?	ASE-SW	n/a	
1.2.6	Are issue papers or national policy required for any of the additional considerations?	ASE-SW	n/a	
1.2.7	Have all non-US certification issues been addressed (if appropriate)?	ASE-SW	A-1, #4	
1.3	Review PSAC and consider the following questions:			
1.3.1	Does the PSAC adequately address the proposed contents described in DO-178B, Section 11.1. If not, are the contents included in another plan?	*ASE-SW/ ASI-SW	A-1, #1-7 A-10, #2	
1.3.2	Is a process in place to address changes that may occur throughout the development process? (This could include revision of PSAC, submittal of a letter summarizing the change and requesting FAA concurrence, etc.)	ASE-SW	A-1, #1,2	
1.3.3	Does the safety assessment adequately support the software level proposed in the PSAC? If the software level is lower than what the safety assessment suggests, is there adequate justification (e.g., through system architecture, partitioning)?	ASE-SW	A-1, #1-4	
1.4	Review SDP and consider the following questions:			

Item #	Evaluation Activity/Question	ASE-SW/ ASI-SW	DO-178B objective	Reviewed? (✓) Issue? (✓*)
1.4.1	If the SDP is followed, will the DO-178B objectives defined in Table A-2 be met.	ASE-SW	A-2, #1-7	
1.4.2	Does the SDP adequately address the proposed contents described in DO-178B, Section 11.2? If not, are the contents included in another plan?	ASE-SW	A-1, #1-4	
1.4.3	Has the software development environment been adequately defined (e.g., compiler options, developmental tools)?	ASE-SW	A-1, #3	
1.4.4	Have the compiler options been identified? (Note: Changes to compiler options may invalidate previous tests and coverage analysis.)	ASE-SW	A-1, #3	
1.4.5	Is the programming language and operating system specified and will they meet the objectives of DO-178B? (Note: Some language and operating system choices may produce non-deterministic results and therefore may not meet the objectives of DO-178B.)	ASE-SW	A-1, #3	
1.5	Review the SCM plan and consider the following questions:			
1.5.1	If the SCM plan is followed, will the DO-178B objectives defined in Table A-8 be met?	ASI-SW	A-8, #1-6	
1.5.2	Are the CM processes described in Section 7.0 of DO-178B in sufficient detail (ref 178B Sections 11.4) to satisfy Section 4.2?	*ASI-SW/ ASE-SW	A-8, #1-6	
1.5.3	Does the SCM plan adequately address the proposed contents described in DO-178B, Section 11.4? If not, are the contents included in another plan?	*ASI-SW/ ASE-SW	A-8, #1-6	

Item #	Evaluation Activity/Question	ASE-SW/ ASI-SW	DO-178B objective	Reviewed? (✓) Issue? (✓*)
1.5.4	<p>Does the SCM plan provide for the following items?</p> <ul style="list-style-type: none"> • Configuration identification of software life cycle data. • Baselineing of all configuration control 1 (CC1) data. • Problem reporting, change control, and configuration status accounting. • Archival, retrieval, and release. • Data retention provisions supporting airworthiness requirements. • Software load control and part numbering to include any additional considerations required for electronic part numbering. • Configuration management of the software life cycle development environment includes tools. • All DO-178B life cycle data to be maintained consistently with the configuration control category associated with the software level. 	ASI-SW	A-8, #1-6	
1.6	Review the SQA plan and consider the following questions:			
1.6.1	If the SQA plan is followed, will the DO-178B objectives defined in Table A-9 be met?	*ASI-SW/ ASE-SW	A-9, #1	
1.6.2	Are the QA integral processes described in Section 8.0 of DO-178B in sufficient detail (ref 178B Section 11.5) to satisfy Section 4.2?	ASI-SW	A1, #1	
1.6.3	Does the SQA plan adequately address the proposed contents described in DO-178B, Section 11.5? If not, are the contents included in another plan?	*ASI-SW/ ASE-SW	A-1, #1	
1.6.4	Are the transition criteria, interrelationships and sequences among process properly and adequately defined?	*ASE-SW/ ASI-SW	A-1, #2; A-9, #2	

Item #	Evaluation Activity/Question	ASE-SW/ ASI-SW	DO-178B objective	Reviewed? (✓) Issue? (✓*)
1.6.5	Has an accountable person/organization been identified for each documented process and activity?	*ASI-SW/ ASE-SW	A-1, #1	
1.7	Review the SVP and consider the following questions:			
1.7.1	If SVP is followed, will objectives of A-3, A-4, A-5, A-6, and A-7 be met?	ASE-SW	A-3 to A-7 (all objectives)	
1.7.2	Does the SVP adequately address the proposed contents described in DO-178B, Section 11.3? If not, are the contents included in another plan?	*ASE-SW/ ASI-SW	A-1, #1-3	
1.7.3	Does the SVP describe how independence will be achieved, when required?	*ASE-SW/ ASI-SW	A-3 to A-7 (all objectives)	
1.7.4	Does the SVP describe the verification method used for each software verification activity?	*ASE-SW/ ASI-SW	A-1, #1-3	
1.7.5	Does the SVP describe the verification environment, including the test equipment? Are there any automated tools? Is there any overlap between various kinds of testing (e.g., overlap of structural and requirements-based tests)? Is the division of the testing task between suppliers and sub-contract suppliers adequately addressed and controlled?	*ASE-SW/ ASI-SW	A-1, #1-3	
1.7.6	Does the SVP describe methods for test case selection?	*ASE-SW/ ASI-SW	A-1, #1-3	
1.8	Develop an understanding of the system from applicant's plans, safety assessment, standards, and briefings.			
1.8.1	Does the safety assessment support the software level for every software component, as proposed in the plans?	ASE-SW	A-1, #1	

Item #	Evaluation Activity/Question	ASE-SW/ ASI-SW	DO-178B objective	Reviewed? (✓) Issue? (✓*)
1.9	Review the software development standards and consider the following questions:			
1.9.1	Have standards been verified for each defined software life cycle process? Are the standards adequate to support the software level?	ASE-SW	A-1, #5	
1.9.2	Have standards been verified to ensure compliance to Section 11?	ASE-SW	A-1, #5	
1.9.3	Have standards been verified to ensure it does not permit any constructs which would invalidate the assumptions about the safety levels (e.g., unconstrained recursion, non-determinism)?	ASE-SW	A-1, #5	

EXAMPLE COMPLIANCE TABLES FROM JOB AID

Anx	Objective	Software Planning: Summary of Compliance Findings/Observations—Level ____ (Date: _____)	Applicable Level	Job Aid
Ref	Summary (Numbers are DO-178B	Applicant:		Ref
#	section references)	Project #:		
		System:		
1-1	Software development and integral processes activities are defined. 4.1 a, 4.3		A/B/C/D	1.1, 1.3, 1.4, 1.6, 1.7, 1.8
1-2	Transition criteria, inter-relationships and sequencing among processes are defined. 4.1b, 4.3		A/B/C	1.1, 1.3, 1.4, 1.6, 1.7
1-3	Software life cycle environment is defined. 4.1c		A/B/C	1.1, 1.2, 1.3, 1.4, 1.7
1-4	Additional considerations are addressed. 4.1d		A/B/C/D	1.1, 1.2, 1.3, 1.4, 2.4
1-5	Software development standards are defined. 4.1e		A/B/C	1.1, 1.3, 1.9
1-6	Software plans comply with this document. 4.1f, 4.6		A/B/C	1.1, 1.3
1-7	Software plans are coordinated. 4.1g, 4.6		A/B/C	1.1, 1.3

Anx	Objective	Software Development: Summary of Compliance Findings/Observations—Level ____ (Date:_____)	Applicable	Job
Ref	Summary (Numbers are DO-178B section references)	Applicant: Project #: System:		Ref
#				
2-1	High-level requirements are developed. 5.1.1a		A/B/C/D	1.5
2-2	Derived high-level requirements are defined. 5.1.1b		A/B/C/D	1.5, 2.1
2-3	Software architecture is developed. 5.2.1a		A/B/C/D	1.5
2-4	Low-level requirements are developed. 5.2.1a		A/B/C/D	1.5
2-5	Derived low-level requirements are defined. 5.2.1b		A/B/C/D	1.5
2-6	Source Code is developed. 5.3.1a		A/B/C/D	1.5
2-7	Executable Object Code is produced and integrated in the target computer. 5.4.1a		A/B/C/D	1.5

Anx	Objective	Verification of Outputs of Software Requirements Process: Summary of Compliance Findings/Observations—Level ____ (Date:_____)	Applicable	Job
Ref #	Summary (Numbers are DO-178B section references)	Applicant: Project #: System:		Ref
3-1	Software high-level requirements comply with system requirements. 6.3.1a		A/B/C/D	2.1
3-2	High-level requirements are accurate and consistent. 6.3.1b		A/B/C/D	2.1
3-3	High-level requirements are compatible with target computer. 6.3.1c		A/B	2.3
3-4	High-level requirements are verifiable. 6.3.1d		A/B/C	2.1
3-5	High-level requirements conform to standards. 6.3.1e		A/B/C	2.1
3-6	High-level requirements are traceable to system requirements. 6.3.1f		A/B/C/D	2.1
3-7	Algorithms are accurate. 6.3.1g		A/B/C	2.1

Anx	Objective	Verification of Outputs of Software Design Process: Summary of Compliance Findings/Observations—Level ____ (Date:_____)	Applicable	Job
Ref #	Summary (Numbers are DO-178B section references)	Applicant: Project #: System:		Ref
4-1	Low-level requirements comply with high-level requirements. 6.3.2a		A/B/C	2.1
4-2	Low-level requirements are accurate and consistent. 6.3.2b		A/B/C	2.1, 2.2
4-3	Low-level requirements are compatible with target computer. 6.3.2c		A/B	2.1
4-4	Low-level requirements are verifiable. 6.3.2d		A/B	2.1, 2.2
4-5	Low-level requirements conform to standards. 6.3.2e		A/B/C	2.1, 2.2
4-6	Low-level requirements are traceable to high-level requirements. 6.3.2f		A/B/C	2.1, 2.2
4-7	Algorithms are accurate. 6.3.2g		A/B/C	2.1, 2.2
4-8	Software architecture is compatible with high-level requirements. 6.3.3a		A/B/C	2.1, 2.3
4-9	Software architecture is consistent. 6.3.3b		A/B/C	2.1, 2.3
4-10	Software architecture is compatible with target computer. 6.2.3c		A/B	2.1, 2.2, 2.3
4-11	Software architecture is verifiable. 6.3.3d		A/B	2.1, 2.3
4-12	Software architecture conforms to standards. 6.3.3e		A/B/C	2.1, 2.2, 2.3
4-13	Software partitioning integrity is confirmed. 6.3.3f		A/B/C/D	2.1, 2.3

Anx	Objective	Verification of Outputs of Software Coding & Integration Process: Summary of Compliance Findings/Observations—Level ____ (Date:_____)	Applicable Level	Job Aid
Ref #	Summary (Numbers are DO-178B section references)	Applicant: Project #: System:		Ref
5-1	Source Code complies with low-level requirements. 6.3.4a		A/B/C	2.4
5-2	Source Code complies with software architecture. 6.3.4b		A/B/C	2.4
5-3	Source Code is verifiable. 6.3.4c		A/B	2.4
5-4	Source Code conforms to standards. 6.3.4d		A/B/C	2.4
5-5	Source Code is traceable to low-level requirements. 6.3.4e		A/B/C	2.4
5-6	Source Code is accurate and consistent. 6.3.4f		A/B/C	2.4
5-7	Output of software integration process is complete and correct. 6.3.5		A/B/C	2.5

Anx	Objective	Testing of Outputs of Integration Process: Summary of Compliance Findings/Observations—Level ____ (Date:_____)	Applicable Level	Job Aid
Ref #	Summary (Numbers are DO-178B section references)	Applicant: Project #: System:		Ref
6-1	Executable Object Code complies with high-level requirements. 6.4.2.1, 6.4.3		A/B/C/D	2.5
6-2	Executable Object Code is robust with high-level requirements. 6.4.2.2, 6.4.3		A/B/C/D	2.5
6-3	Executable Object Code complies with low-level requirements. 6.4.2.1, 6.4.3		A/B/C	2.5
6-4	Executable Object Code is robust with low-level requirements. 6.4.2.2, 6.4.3		A/B/C	2.5
6-5	Executable Object Code is compatible with target computer. 6.4.3a		A/B/C/D	2.5

Anx	Objective	Verification of Verification Process Results: Summary of Compliance Findings/Observations—Level ____ (Date:_____)	Applicable	Job
Ref #	Summary (Numbers are DO-178B section references)	Applicant: Project #: System:		Ref
7-1	Test procedures are correct. 6.3,6b		A/B/C	2.10, 3.2, 3.3
7-2	Test results are correct and discrepancies explained. 6.3.6c		A/B/C	2.10, 3.3
7-3	Test coverage of high-level requirements is achieved. 6.4.4.1		A/B/C/D	2.10, 3.1, 3.2, 3.3
7-4	Test coverage of low-level requirements is achieved. 6.4.4.2		A/B/C	2.10, 3.2, 3.3
7-5	Test coverage of software structure (modified condition/decision) is achieved. 6.4.4.2		A	2.10, 3.2, 3.3
7-6	Test coverage of software structure (decision coverage) is achieved. 6.4.4.2a, 6.4.4.2b		A/B	2.10, 3.2, 3.3
7-7	Test coverage of software structure (statement coverage) is achieved. 6.4.4.2a, 6.4.4.2b		A/B/C	2.10, 3.2, 3.3
7-8	Test coverage of software structure (data coupling and control coupling) is achieved. 6.4.4.2c		A/B/C	2.10, 3.2, 3.3

Anx	Objective	Software Configuration Management Process: Summary of Compliance Findings/Observations—Level ____ (Date:_____)	Applicable	Job
Ref #	Summary (Numbers are DO-178B section references)	Applicant: Project #: System:		Ref
8-1	Configuration items are identified. 7.2.1		A/B/C/D	1.5, 2.6, 3.4
8-2	Baselines and traceability are established. 7.2.2		A/B/C/D	1.5, 2.6, 3.4
8-3	Problem reporting, change control, change review, and configuration status accounting are established. 7.2.3, 7.2.4, 7.2.5, 7.2.6		A/B/C/D	1.5, 2.4, 2.7, 3.5
8-4	Archive, retrieval, and release are established. 7.2.7		A/B/C/D	1.5, 2.8, 3.6
8-5	Software load control is established. 7.2.8		A/B/C/D	1.5, 3.8
8-6	Software life cycle environment control is established. 7.2.9		A/B/C/D	1.5, 3.2

Anx	Objective	Software Quality Assurance Process: Summary of Compliance Findings/Observations—Level ____ (Date:_____)	Applicable	Job
Ref #	Summary (Numbers are DO-178B section references)	Applicant: Project #: System:		Ref
9-1	Assurance is obtained that software development and integral processes comply with approved software plans and standards. 8.1a		A/B/C/D	1.6, 2.1, 2.2, 2.9, 3.7
9-2	Assurance is obtained that transition criteria for the software life cycle processes are satisfied. 8.1b		A/B	1.6, 2.9, 3.2
9-3	Software conformity review is conducted. 8.1c, 8.3		A/B/C/D	2.9

Anx	Objective	Certification Liaison Process: Summary of Compliance Findings/Observations—Level ____ (Date:_____)	Applicable	Job
Ref #	Summary (Numbers are DO-178B section references)	Applicant: Project #: System:		Ref
10-1	Communication and understanding between the applicant and the certification authority is established. 9.0		A/B/C/D	1.1 - 1.9
10-2	The means of compliance is proposed and agreement with the Plan for Software Aspects of Certification is obtained. 9.1		A/B/C/D	1.3
10-3	Compliance substantiation is provided. 9.2		A/B/C/D	4.1-4.8

DO-178B Planning Information

Appendix F

11.1

Plan for Software Aspects of Certification

The Plan for Software Aspects of Certification is the primary means used by the certification authority for determining whether an applicant is proposing a software life cycle that is commensurate with the rigor required for the level of software being developed. This plan should include:

- a. System overview: This section provides an overview of the system, including a description of its functions and their allocation to the hardware and software, the architecture, processor(s) used, hardware/software interfaces, and safety features.
- b. Software overview: This section briefly describes the software functions with emphasis on the proposed safety and partitioning concepts, for example, resource sharing, redundancy, multiple-version dissimilar software, fault tolerance, and timing and scheduling strategies.
- c. Certification considerations: This section provides a summary of the certification basis, including the means of compliance, as relating to the software aspects of certification. This section also states the proposed software level(s) and summarizes the justification provided by the system safety assessment process, including potential software contributions to failure conditions.
- d. Software life cycle: This section defines the software life cycle to be used and includes a summary of each software life cycle and its processes for which detailed information is defined in their respective software plans. The summary explains how the objectives of each software life cycle process will be satisfied, and specifies the organizations to be involved, the organizational responsibilities, and the system life cycle processes and certification liaison process responsibilities.
- e. Software life cycle data: This section specifies the software life cycle data that will be produced and controlled by the software life cycle processes. This section also describes the relationship of the data to each other or to other data defining the system, the software life cycle data to be submitted to the certification authority, the form of the data, and the means by which software life cycle data will be made available to the certification authority.
- f. Schedule: This section describes the means the applicant will use to provide the certification authority with visibility of the activities of the software life cycle processes so reviews can be planned.
- g. Additional considerations: This section describes specific features that may affect the certification process, for example, alternative methods of compliance, tool qualification, previously developed software, option-selectable software, user-modifiable software, COTS software, field-loadable software, multiple-version dissimilar software, and product service history.

11.2

Software Development Plan

The Software Development Plan includes the objectives, standards and software life cycle(s) to be used in the software development processes. It may be included in the Plan for Software Aspects of Certification. This plan should include:

- a. Standards: Identification of the Software Requirements Standards, Software Design Standards and Software Code Standards for the project. Also, references to the standards for previously developed software, including COTS software, if those standards are different.
- b. Software life cycle: A description of the software life cycle processes to be used to form the specific software life cycle(s) to be used on the project, including the

transition criteria for the software development processes. This description is distinct from the summary provided in the Plan for Software Aspects of Certification, in that it provides the detail necessary to ensure proper implementation of the software life cycle processes.

- c. Software development environment: A statement of the chosen software development environment in terms of hardware and software, including:
 - (1) The chosen requirements development method(s) and tools to be used.
 - (2) The chosen design method(s) and tools to be used.
 - (3) The programming language(s), coding tools, compilers, linkage editors and loaders to be used.
 - (4) The hardware platforms for the tools to be used.

11.3

Software Verification Plan

The Software Verification Plan is a description of the verification procedures to satisfy the software verification process objectives. These procedures may vary by software level as defined in the tables of Annex A. This plan should include:

- a. Organization: Organizational responsibilities within the software verification process and interfaces with the other software life cycle processes.
- b. Independence: A description of the methods for establishing verification independence, when required.
- c. Verification methods: A description of the verification methods to be used for each activity of the software verification process.
 - (1) Review methods, including checklists or other aids.
 - (2) Analysis methods, including traceability and coverage analysis.
 - (3) Testing methods, including guidelines that establish the test case selection process, the test procedures to be used, and the test data to be produced.
- d. Verification environment: A description of the equipment for testing, the testing and analysis tools, and the guidelines for applying these tools and hardware test equipment (see also paragraph 4.4.3, item b for guidance on indicating target computer and simulator or emulator differences).
- e. Transition criteria: The transition criteria for entering the software verification process defined in this plan.
- f. Partitioning considerations: If partitioning is used, the methods used to verify the integrity of the partitioning.
- g. Compiler assumptions: A description of the assumptions made by the applicant about the correctness of the compiler, linkage editor or loader (paragraph 4.4.2).
- h. Reverification guidelines: For software modification, a description of the methods for identifying the affected areas of the software and the changed parts of the Executable Object Code. The reverification should ensure that previously reported errors or classes of errors have been eliminated.
- i. Previously developed software: For previously developed software, if the initial compliance baseline for the verification process does not comply with this document, a description of the methods to satisfy the objectives of this document.

- j. Multiple-version dissimilar software: If multiple-version dissimilar software is used, a description of the software verification process activities (paragraph 12.3.3).

11.4

Software Configuration Management Plan

The Software Configuration Management Plan establishes the methods to be used to achieve the objectives of the software configuration management (SCM) process throughout the software life cycle. This plan should include:

- a. Environment: A description of the SCM environment to be used, including procedures, tools, methods, standards, organizational responsibilities, and interfaces.
- b. Activities: A description of the SCM process activities in the software life cycle that will satisfy the objectives for:
 - (1) Configuration identification: Items to be identified, when they will be identified, the identification methods for software life cycle data (for example, part numbering), and the relationship of software identification and airborne system or equipment identification.
 - (2) Baselines and traceability: The means of establishing baselines, what baselines will be established, when these baselines will be established, the software library controls, and the configuration item and baseline traceability.
 - (3) Problem reporting: The content and identification of problem reports for the software product and software life cycle processes, when they will be written, the method of closing problem reports, and the relationship to the change control activity.
 - (4) Change control: Configuration items and baselines to be controlled, when they will be controlled, the problem/change control activities that control them, pre-certification controls, post-certification controls, and the means of preserving the integrity of baselines and configuration items.
 - (5) Change review: The method of handling feedback from and to the software life cycle processes; the methods of assessing and prioritizing problems, approving changes, and handling their resolution or change implementation; and the relationship of these methods to the problem reporting and change control activities.
 - (6) Configuration status accounting: The data to be recorded to enable reporting configuration management status, definition of where that data will be kept, how it will be retrieved for reporting, and when it will be available.
 - (7) Archive, retrieval, and release: The integrity controls, the release method and authority, and data retention.
 - (8) Software load control: A description of the software load control safeguards and records.
 - (9) Software life cycle environment controls: Controls for the tools used to develop, build, verify and load the software, addressing items 1 through 7 above. This includes control of tools to be qualified.
 - (10) Software life cycle data controls: Controls associated with Control Category 1 and Control Category 2 data.
- c. Transition criteria: The transition criteria for entering the SCM process.

- d. SCM data: A definition of the software life cycle data produced by the SCM process, including SCM Records, the Software Configuration Index and the Software Life Cycle Environment Configuration Index.
- e. Supplier control: The means of applying SCM process requirements to sub-tier suppliers.

11.5 Software Quality Assurance Plan

The Software Quality Assurance Plan establishes the methods to be used to achieve the objectives of the software quality assurance (SQA) process. The SQA Plan may include descriptions of process improvement, metrics, and progressive management methods. This plan should include:

- a. Environment: A description of the SQA environment, including scope, organizational responsibilities and interfaces, standards, procedures, tools and methods.
- b. Authority: A statement of the SQA authority, responsibility, and independence, including the approval authority for software products.
- c. Activities: The SQA activities that are to be performed for each software life cycle process and throughout the software life cycle including:
 - (1) SQA methods, for example, reviews, audits, reporting, inspections, and monitoring of the software life cycle processes.
 - (2) Activities related to the problem reporting, tracking and corrective action system.
 - (3) A description of the software conformity review activity.
- d. Transition criteria: The transition criteria for entering the SQA process.
- e. Timing: The timing of the SQA process activities in relation to the activities of the software life cycle processes.
- f. SQA Records: A definition of the records to be produced by the SQA process.
- g. Supplier control: A description of the means of ensuring that sub-tier suppliers' processes and outputs will comply with the SQA Plan.

11.6 Software Requirements Standards

The purpose of Software Requirements Standards is to define the methods, rules and tools to be used to develop the high-level requirements. These standards should include:

- a. The methods to be used for developing software requirements, such as structured methods.
- b. Notations to be used to express requirements, such as data flow diagrams and formal specification languages.
- c. Constraints on the use of the requirement development tools.
- d. The method to be used to provide derived requirements to the system process.

11.7 Software Design Standards

The purpose of Software Design Standards is to define the methods, rules and tools to be used to develop the software architecture and low-level requirements. These standards should include:

- a. Design description method(s) to be used.

- b. Naming conventions to be used.
- c. Conditions imposed on permitted design methods, for example, scheduling, and the use of interrupts and event-driven architectures, dynamic tasking, re-entry, global data, and exception handling, and rationale for their use.
- d. Constraints on the use of the design tools.
- e. Constraints on design, for example, exclusion of recursion, dynamic objects, data aliases, and compacted expressions.
- f. Complexity restrictions, for example, maximum level of nested calls or conditional structures, use of unconditional branches, and number of entry/exit points of code components.

11.8

Software Code Standards

The purpose of the Software Code Standards is to define the programming languages, methods, rules and tools to be used to code the software. These standards should include:

- a. Programming language(s) to be used and/or defined subset(s). For a programming language, reference the data that unambiguously defines the syntax, the control behavior, the data behavior and side-effects of the language. This may require limiting the use of some features of a language.
 - b. Source Code presentation standards, for example, line length restriction, indentation, and blank line usage and Source Code documentation standards, for example, name of author, revision history, inputs and outputs, and affected global data.
- c. Naming conventions for components, subprograms, variables, and constants.
- d. Conditions and constraints imposed on permitted coding conventions, such as the degree of coupling between software components and the complexity of logical or numerical expressions and rationale for their use.
- e. Constraints on the use of the coding tools.

Example Software Requirements

Appendix G

Requirements for TDLRSP (Touch Down Landing Radar Sensor Processing)

Figure 4.1 DFD 2.1: SP – SENSOR PROCESSING

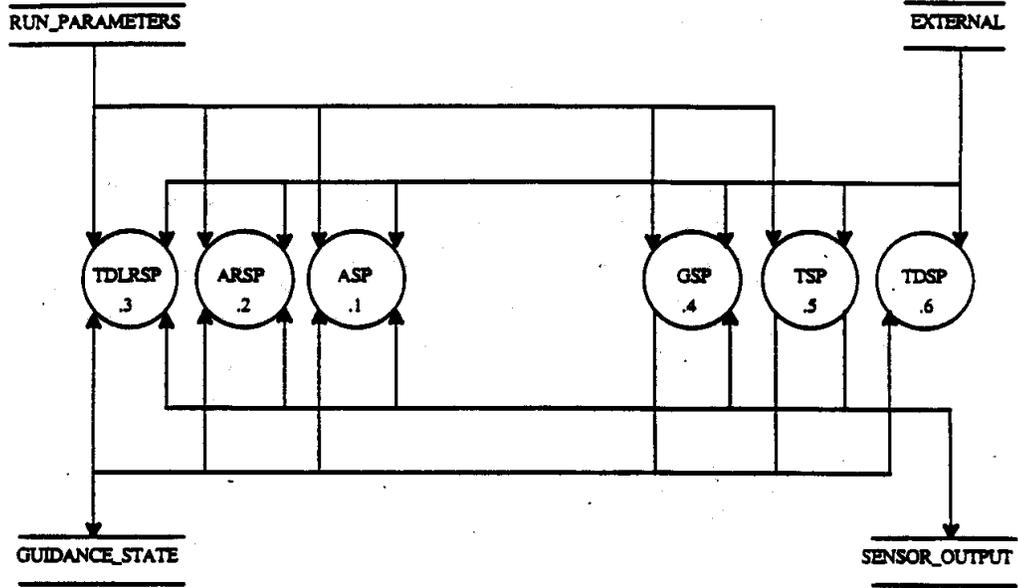


Figure 4.2 CFD 2.1: SP – SENSOR PROCESSING

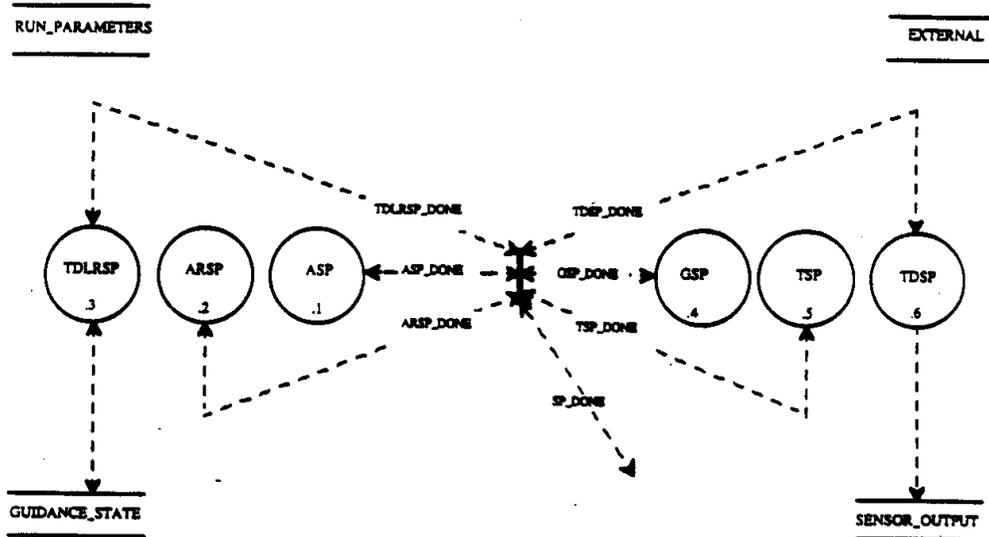


Table 4.1 C-Spec 2.1: SP – SENSOR PROCESSING

	"ASP"	"ARSP"	"TDLRSP"	"GSP"	"TSP"	"TDSP"	ASP_DONE	ARSP_DONE	TDLRSP_DONE	GSP_DONE	TSP_DONE	TDSP_DONE	SP_DONE
~ASP_DONE & ~ARSP_DONE & ~TDLRSP_DONE & ~GSP_DONE & ~TSP_DONE & ~TDSP_DONE & ~SP_DONE	2	2	2	2	1	2							
ASP_DONE & ARSP_DONE & TDLRSP_DONE & GSP_DONE & TSP_DONE & TDSP_DONE & ~SP_DONE							"FALSE"	"FALSE"	"FALSE"	"FALSE"	"FALSE"	"FALSE"	"TRUE"

TDLRSP – Touch Down Landing Radar Sensor Processing (P-Spec 2.1.3)

PURPOSE A single touch down landing radar (TDLR) gauges the velocity of the vehicle during terminal descent. This radar is a doppler radar with four radar beams, each of which emanates from the vehicle's center of gravity with a slight offset from the vehicle's \bar{x} , axis. The radar beams form the edges of the pyramid as shown in Figure 5.3

The Touch Down Landing Radar Sensor Processing (TDLRSP) functional unit converts measurements of the frequency shift of each beams reflection into vehicle velocities; however, the receivers associated with each beam may not find a usable reflection. If no usable reflection is found, the receiver returns a status of beam in search mode (unlocked).

INPUT

DELTA_T	FRAME_BEAM_UNLOCKED
FRAME_COUNTER	K_MATRIX
TDLR_ANGLES	TDLR_COUNTER
TDLR_GAIN	TDLR_LOCK_TIME
TDLR_OFFSET	TDLR_STATE
TDLR_VELOCITY	

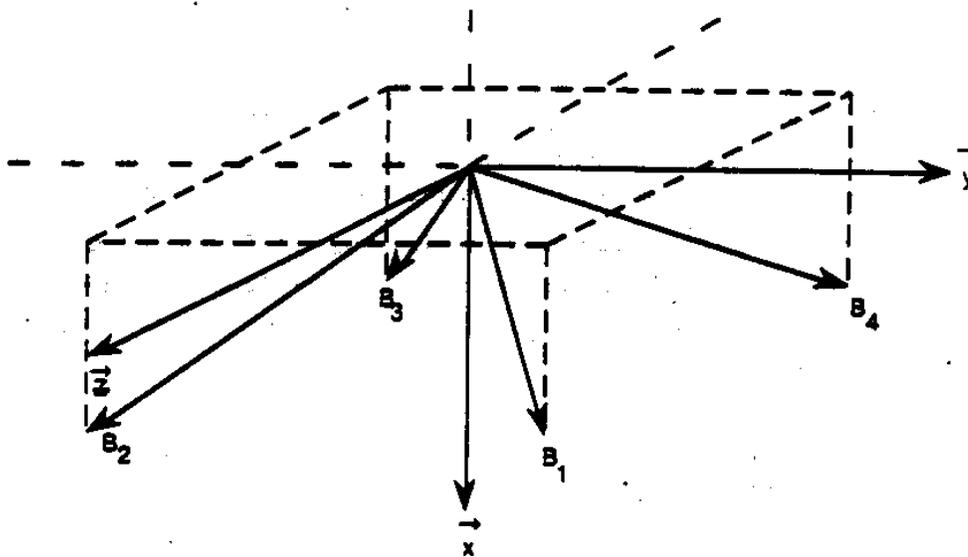
OUTPUT

FRAME_BEAM_UNLOCKED	K_MATRIX
TDLR_STATE	TDLR_STATUS
TDLR_VELOCITY	

PROCESS The value returned by each beam (TDLR_COUNTER) is proportional to the beam frequency shift down that beam, which is, in turn, proportional to the velocity down that beam. The processing of the TDLR_COUNTER data into the component velocities along the vehicle's \bar{x} , \bar{y} , and \bar{z} axes requires the following steps:

- ✓ **ROTATE VARIABLES**
 - Rotate TDLR_VELOCITY and K_MATRIX.

Figure 5.3 DOPPLER RADAR BEAM LOCATIONS



✓ **DETERMINE RADAR BEAM STATES**

The processing of the four radar beams depends on the current state of the radar, i.e. whether or not each of the four beams is searching or in lock, and also upon the previous states of the beams. Note that at the beginning of each trajectory, `FRAME_BEAM_UNLOCKED` will be set to zero, thus meaning that the beam has never been unlocked. If the receiver for a beam does not sense an echo (i.e. the beam is in search mode), the corresponding `TDLR_COUNTER` value will be zero. Note that a beam which becomes unlocked will be ignored for `TDLR_LOCK_TIME` seconds.

- Use Table 5.11 to determine the state (`TDLR_STATE` and `FRAME_BEAM_UNLOCKED`) for each of the four beams.

Table 5.11 DETERMINATION OF RADAR BEAM STATES

CURRENT STATE			ACTIONS	
TDLR_STATE	TDLR_COUNTER	ΔT $(FRAME_COUNTER - FRAME_BEAM_UNLOCKED) \geq TDLR_LOCK_TIME?$	TDLR_STATE	FRAME_BEAM_UNLOCKED
locked	0	d	unlocked	current FRAME_COUNTER
unlocked	$\neq 0$	yes	locked	
unlocked	0	yes		current FRAME_COUNTER

Note: A blank box under "ACTIONS" indicates no action is to be taken
"d" = don't care condition

3

✓ DETERMINE BEAM VELOCITIES

A beam velocity is a linear function of its TDLR_COUNTER value where the gain (TDLR_GAIN) specifies the slope and the offset (TDLR_OFFSET) specifies the intercept.

- Calculate the beam velocities as follows:

$$B(i) = TDLR_OFFSET + TDLR_GAIN * (TDLR_COUNTER(i))$$

where i ranges from 1 to 4 and represents the four radar beams.

4

✓ PROCESS THE BEAM VELOCITIES

- Use Table 5.12 to calculate values for \hat{B}_x , \hat{B}_y , and \hat{B}_z , which are the processed beam velocities. Note that in Table 5.12, B_i is shorthand for $B(i)$, where i ranges from 1 to 4. Note also that the knowledge of which beams are in lock is used to determine which line of the table to use in order to calculate \hat{B}_x , \hat{B}_y , and \hat{B}_z .

5

✓ CONVERT TO BODY VELOCITIES

- In order to convert the processed beam velocities to body velocities (TDLR_VELOCITY), use the following equations, which make use of the angles α , β and γ (TDLR_ANGLES) which are the offsets of the beams from the body axes:

$$TDLR_VELOCITY(1) = \frac{\hat{B}_x}{\cos \alpha}$$

$$TDLR_VELOCITY(2) = \frac{\hat{B}_y}{\cos \beta}$$

$$TDLR_VELOCITY(3) = \frac{\hat{B}_z}{\cos \gamma}$$

✓ **SET VALUES IN K_MATRIX**

When calculating the vehicle velocity, the Guidance Processor must know which components of the body velocities are usable. A value of one in the diagonal element of the K_MATRIX indicates that the corresponding velocity should be used, while a value of zero indicates that it should not.

- Use Table 5.12 to set the values for K_x , K_y , and K_z in K_MATRIX, (again on the basis of which beams are in lock), as follows:

$$K_MATRIX = \begin{pmatrix} K_x & 0 & 0 \\ 0 & K_y & 0 \\ 0 & 0 & K_z \end{pmatrix}$$

The off-diagonal elements of K_MATRIX should not be updated.

✓ **SET TDLR_STATUS**

- Set all elements of TDLR_STATUS to healthy.

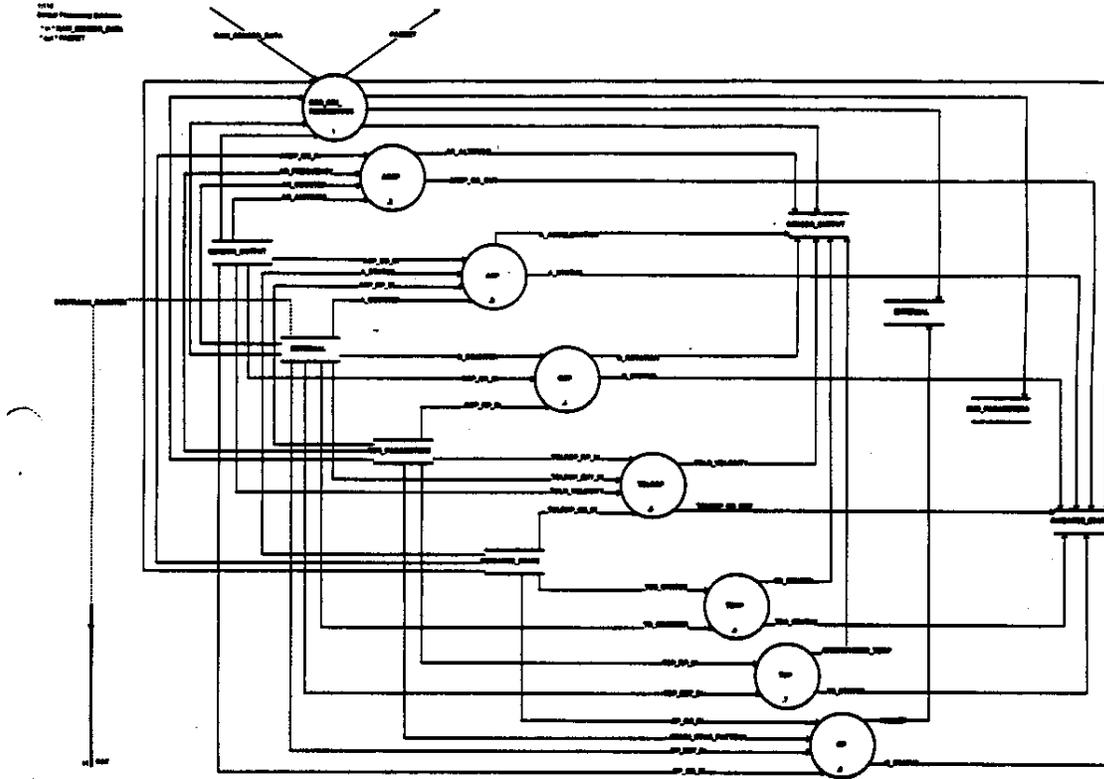
Table 5.12 PROCESSING OF DOPPLER RADAR BEAMS IN LOCK

CURRENT STATE BEAMS IN LOCK	ACTIONS					
	\dot{B}_x	K_x	\dot{B}_y	K_y	\dot{B}_z	K_z
none	0	0	0	0	0	0
B_1	0	0	0	0	0	0
B_2	0	0	0	0	0	0
B_3	0	0	0	0	0	0
B_4	0	0	0	0	0	0
B_1, B_2	0	0	$(B_1 - B_2)/2$	1	0	0
B_1, B_3	$(B_1 + B_3)/2$	1	0	0	0	0
B_1, B_4	0	0	0	0	$(B_1 - B_4)/2$	1
B_2, B_3	0	0	0	0	$(B_2 - B_3)/2$	1
B_2, B_4	$(B_2 + B_4)/2$	1	0	0	0	0
B_3, B_4	0	0	$(B_4 - B_3)/2$	1	0	0
B_1, B_2, B_3	$(B_1 + B_3)/2$	1	$(B_1 - B_2)/2$	1	$(B_2 - B_3)/2$	1
B_1, B_2, B_4	$(B_2 + B_4)/2$	1	$(B_1 - B_2)/2$	1	$(B_1 - B_4)/2$	1
B_1, B_3, B_4	$(B_1 + B_3)/2$	1	$(B_4 - B_3)/2$	1	$(B_1 - B_4)/2$	1
B_2, B_3, B_4	$(B_2 + B_4)/2$	1	$(B_4 - B_3)/2$	1	$(B_2 - B_3)/2$	1
B_1, B_2, B_3, B_4	$(B_1 + B_2 + B_3 + B_4)/4$	1	$(B_1 - B_2 - B_3 + B_4)/4$	1	$(B_1 + B_2 - B_3 - B_4)/4$	1

Example Software Design

Appendix H

Design for TDLRSP (Touch Down Landing Radar Sensor Processing)



NAME:
1.5;27

TITLE:
TDLRSP

INPUT/OUTPUT:
DELTA_T : data_in
FRAME_BEAM_UNLOCKED : data_in
FRAME_COUNTER : data_in
K_MATRIX : data_in
TDLR_ANGLES : data_in
TDLR_COUNTER : data_in
TDLR_GAIN : data_in
TDLR_LOCK_TIME : data_in
TDLR_OFFSET : data_in

TDLR_STATE : data_in

TDLR_VELOCITY : data_in
FRAME_BEAM_UNLOCKED : data_out

K_MATRIX : data_out
TDLR_STATE : data_out

TDLR_STATUS : data_out

TDLR_VELOCITY : data_out

BODY:
BEGIN P_SPEC

```
(*****  
* TDLRSP -- Touch Down Landing Radar Sensor Processing  
*  
* TDLRSP processing is responsible for:  
* 1) Maintaining the history of the vehicle velocities and the  
* velocity computation indicator,  
* 2) Determining the operational status of touch down landing radar  
* sensor, and  
* 3) Reporting the current vehicle velocities along each of the  
* vehicle's three axes, and  
* 4) Reporting the velocity computation indicators.  
*  
*****)
```

```
(*****  
* 1) Maintain the history of the vehicle velocities and the  
* velocity computation indicator by "rotating variables." The data  
* element TDLR_VELOCITY is defined as a two dimensional array. The  
* first dimension represents a vehicle axis: x-axis (1), y-axis  
* (2), and z-axis (3). The second dimension represents a five deep  
* history. The data element K_MATRIX is defined as a three  
* dimensional array (1..3, 1..3, 0..4). The velocity computation  
* indicators are arranged as a 3x3 matrix, represented by the first
```

- * two dimensions of K_MATRIX. The third dimension represents a
 - * five deep history. The first element of the history, element
 - * zero, holds the most recently computed value. The last element
 - * of the history, element four, holds the oldest maintained value.
 - * In shifting the values stored in these data elements, a
 - * multi-frame history is maintained.
- *****)

```

TDLR_VELOCITY[1, 4] := TDLR_VELOCITY[1, 3]
TDLR_VELOCITY[1, 3] := TDLR_VELOCITY[1, 2]
TDLR_VELOCITY[1, 2] := TDLR_VELOCITY[1, 1]
TDLR_VELOCITY[1, 1] := TDLR_VELOCITY[1, 0]

```

```

TDLR_VELOCITY[2, 4] := TDLR_VELOCITY[2, 3]
TDLR_VELOCITY[2, 3] := TDLR_VELOCITY[2, 2]
TDLR_VELOCITY[2, 2] := TDLR_VELOCITY[2, 1]
TDLR_VELOCITY[2, 1] := TDLR_VELOCITY[2, 0]

```

```

TDLR_VELOCITY[3, 4] := TDLR_VELOCITY[3, 3]
TDLR_VELOCITY[3, 3] := TDLR_VELOCITY[3, 2]
TDLR_VELOCITY[3, 2] := TDLR_VELOCITY[3, 1]
TDLR_VELOCITY[3, 1] := TDLR_VELOCITY[3, 0]

```

```

K_MATRIX[1, 1, 4] := K_MATRIX[1, 1, 3]
K_MATRIX[1, 2, 4] := K_MATRIX[1, 2, 3]
K_MATRIX[1, 3, 4] := K_MATRIX[1, 3, 3]
K_MATRIX[2, 1, 4] := K_MATRIX[2, 1, 3]
K_MATRIX[2, 2, 4] := K_MATRIX[2, 2, 3]
K_MATRIX[2, 3, 4] := K_MATRIX[2, 3, 3]
K_MATRIX[3, 1, 4] := K_MATRIX[3, 1, 3]
K_MATRIX[3, 2, 4] := K_MATRIX[3, 2, 3]
K_MATRIX[3, 3, 4] := K_MATRIX[3, 3, 3]

```

```

K_MATRIX[1, 1, 3] := K_MATRIX[1, 1, 2]
K_MATRIX[1, 2, 3] := K_MATRIX[1, 2, 2]
K_MATRIX[1, 3, 3] := K_MATRIX[1, 3, 2]
K_MATRIX[2, 1, 3] := K_MATRIX[2, 1, 2]
K_MATRIX[2, 2, 3] := K_MATRIX[2, 2, 2]
K_MATRIX[2, 3, 3] := K_MATRIX[2, 3, 2]
K_MATRIX[3, 1, 3] := K_MATRIX[3, 1, 2]
K_MATRIX[3, 2, 3] := K_MATRIX[3, 2, 2]
K_MATRIX[3, 3, 3] := K_MATRIX[3, 3, 2]

```

```

K_MATRIX[1, 1, 2] := K_MATRIX[1, 1, 1]
K_MATRIX[1, 2, 2] := K_MATRIX[1, 2, 1]
K_MATRIX[1, 3, 2] := K_MATRIX[1, 3, 1]
K_MATRIX[2, 1, 2] := K_MATRIX[2, 1, 1]
K_MATRIX[2, 2, 2] := K_MATRIX[2, 2, 1]
K_MATRIX[2, 3, 2] := K_MATRIX[2, 3, 1]
K_MATRIX[3, 1, 2] := K_MATRIX[3, 1, 1]
K_MATRIX[3, 2, 2] := K_MATRIX[3, 2, 1]
K_MATRIX[3, 3, 2] := K_MATRIX[3, 3, 1]

```

```

K_MATRIX[1, 1, 1] := K_MATRIX[1, 1, 0]
K_MATRIX[1, 2, 1] := K_MATRIX[1, 2, 0]
K_MATRIX[1, 3, 1] := K_MATRIX[1, 3, 0]

```

```

K_MATRIX[2, 1, 1] := K_MATRIX[2, 1, 0]
K_MATRIX[2, 2, 1] := K_MATRIX[2, 2, 0]
K_MATRIX[2, 3, 1] := K_MATRIX[2, 3, 0]
K_MATRIX[3, 1, 1] := K_MATRIX[3, 1, 0]
K_MATRIX[3, 2, 1] := K_MATRIX[3, 2, 0]
K_MATRIX[3, 3, 1] := K_MATRIX[3, 3, 0]

(*****
* 2) Determine the operational status of touch down landing radar
* sensor.
*
* The operational status of the TDLR sensor is always reported
* as "healthy" (value 0).
*****)

TDLR_STATUS[1] := 0
TDLR_STATUS[2] := 0
TDLR_STATUS[3] := 0
TDLR_STATUS[4] := 0

(*****
* 3) Reporting the current vehicle velocities along each of the
* vehicle's three axes and reporting the velocity computation
* indicators.
*****)

(*****
* 3A) Determine the state of the four radar beams.
*
* The data element TDLR_STATE contains the state of the radar
* beams.
*
* Valid radar beam states are "locked" (value 1) and "unlocked"
* (value 0). The present state of a radar beam is determined from
* the current value of the sensor data and the previous state of
* the radar beam. A sensor measurement of zero indicates that the
* radar beam echo was not received and the radar beam is considered
* to be "unlocked." A non-zero sensor measurement indicates that a
* radar beam echo was received, but does not imply a radar beam
* state of "locked." Because, once a radar beam is declared
* "unlocked," it is rendered unusable (remains "unlocked"
* regardless of the sensor data value) for a specified period of
* time. This waiting period must be implemented in the software.
*
* A beam is deemed "locked" when 1) the current sensor value
* contains a non-zero value and the beam's previous state was
* "locked"; or 2) the current sensor value contains a non-zero
* value and the beam's previous state was "unlocked" and the
* elapsed time since the beam was determined "unlocked" is greater
* than or equal to the sensor recovery period.
*
* The data element TDLR_LOCK_TIME specifies the unlocked sensor
* recovery (waiting) period. The data element FRAME_BEAM_UNLOCKED
* is updated with the value of the FRAME_COUNTER during the frame
* in which a radar beam state is first determined as "unlocked."
* The data element DELTA_T specifies in seconds the duration of a

```

```

* single frame. Thus the elapsed time since a radar beam was
* declared "unlocked" can be determined by subtracting the present
* value of FRAME_COUNTER from the value of FRAME_BEAM_UNLOCKED and
* multiplying the result by the value of DELTA_T.
*****

do (for each radar beam i :=(1 to 4))                                     (1)

    if (TDLR_COUNTER[i] = 0) then      (* beam is unlocked *)           (2)

        if (TDLR_STATE[i] = 1) then    (* beam was locked *)           (3)
            TDLR_STATE[i] := 0        (* set unlocked *)
            FRAME_BEAM_UNLOCKED[i] := FRAME_COUNTER

        else      (* the beam was unlocked *)                           (3)

            elapsed_time := DELTA_T * (FRAME_COUNTER - FRAME_BEAM_UNLOCKED[i])

            if (elapsed_time >= TDLR_LOCK_TIME) then                    (4)
                FRAME_BEAM_UNLOCKED[i] := FRAME_COUNTER
            end if                                                       (4)

        end if                                                           (3)

    else      (* the sensor measurement != 0 *)                         (2)

        if (TDLR_STATE[i] = 0) then    (* beam was unlocked *)         (3)

            elapsed_time := DELTA_T * (FRAME_COUNTER - FRAME_BEAM_UNLOCKED[i])

            if (elapsed_time >= TDLR_LOCK_TIME) then                    (4)
                TDLR_STATE[i] := 1      (* set locked *)
            end if                                                       (4)

        end if                                                           (3)

    end if                                                                (2)

end do (* for each beam i *)                                           (1)

*****
* 3B) Determine the beam velocities.
*****

do (for each radar beam i := (1 to 4))
    b[i] := TDLR_OFFSET + TDLR_GAIN * TDLR_COUNTER[i]
end do (* for each beam *)

*****
* 3C) Determine the "processed" beam velocities, and
* 4) Determine the velocity computation indicators.
*****
*****
* Compute a "processed" beam velocity for each of the three axes as
* specified by the following table:

```

```

*
* Beams | PROCESSED BEAM VELOCITIES | K-MATRIX | Case
* in lock | pbvX | pbvY | pbvZ | X | Y | Z | Number
* -----|-----|-----|-----|-----|-----|-----|-----
* none | 0 | 0 | 0 | 0 | 0 | 0 | 0
* 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1
* 2 | 0 | 0 | 0 | 0 | 0 | 0 | 2
* 3 | 0 | 0 | 0 | 0 | 0 | 0 | 4
* 4 | 0 | 0 | 0 | 0 | 0 | 0 | 8
* -----|-----|-----|-----|-----|-----|-----|-----
* 1,2 | 0 | (b[1]-b[2])/2 | 0 | 0 | 1 | 0 | 3
* 1,3 | (b[1]+b[3])/2 | 0 | 0 | 1 | 0 | 0 | 5
* 1,4 | 0 | 0 | (b[1]-b[4])/2 | 0 | 0 | 1 | 9
* 2,3 | 0 | 0 | (b[2]-b[3])/2 | 0 | 0 | 1 | 6
* 2,4 | (b[2]+b[4])/2 | 0 | 0 | 1 | 0 | 0 | 10
* 3,4 | 0 | (b[4]-b[3])/2 | 0 | 0 | 1 | 0 | 12
* -----|-----|-----|-----|-----|-----|-----|-----
* 1,2,3 | (b[1]+b[3])/2 | (b[1]-b[2])/2 | (b[2]-b[3])/2 | 1 | 1 | 1 | 7
* 1,2,4 | (b[2]+b[4])/2 | (b[1]-b[2])/2 | (b[1]-b[4])/2 | 1 | 1 | 1 | 11
* 1,3,4 | (b[1]+b[3])/2 | (b[4]-b[3])/2 | (b[1]-b[4])/2 | 1 | 1 | 1 | 13
* 2,3,4 | (b[2]+b[4])/2 | (b[4]-b[3])/2 | (b[2]-b[3])/2 | 1 | 1 | 1 | 14
* -----|-----|-----|-----|-----|-----|-----|-----
* 1,2,3,4 | a | b | c | 1 | 1 | 1 | 15
*
* a) (b[1]+b[2]+b[3]+b[4])/4
* b) (b[1]-b[2]-b[3]+b[4])/4
* c) (b[1]+b[2]-b[3]-b[4])/4
*
* Each of the 16 possible cases has been assigned a case number to
* facilitate the description of the necessary processing. The case
* number is found in the column labied "Case Number" in the table
* above.
*
* Determine the case number value for the current processing.
* Each of the four radar beams' state has been assigned a weight
* value: beam 1: 1, beam 2: 2, beam 3: 4, beam 4: 8. The "case
* number" is computed by summing the radar beams multiplied by their
* their weight factors.
*****

```

```

state_case := TDLR_STATE[1] + 2*TDLR_STATE[2] +
              4*TDLR_STATE[3] + 8*TDLR_STATE[4]

```

```

case state_case of
  0, 1, 2, 4, 8:
    pbvX := 0
    pbvY := 0
    pbvZ := 0

    K_MATRIX[1, 1, 0] := 0
    K_MATRIX[2, 2, 0] := 0
    K_MATRIX[3, 3, 0] := 0
  end

  3: pbvX := 0
     pbvY := (b1-b2)/2

```

```

        pbvZ := 0

        K_MATRIX[1, 1, 0] := 0
        K_MATRIX[2, 2, 0] := 1
        K_MATRIX[3, 3, 0] := 0
    end

5:   pbvX := (b1+b3)/2
    pbvY := 0
    pbvZ := 0

        K_MATRIX[1, 1, 0] := 1
        K_MATRIX[2, 2, 0] := 0
        K_MATRIX[3, 3, 0] := 0
    end

9:   pbvX := 0
    pbvY := 0
    pbvZ := (b1-b4)/2

        K_MATRIX[1, 1, 0] := 0
        K_MATRIX[2, 2, 0] := 0
        K_MATRIX[3, 3, 0] := 1
    end

6:   pbvX := 0
    pbvY := 0
    pbvZ := (b2-b3)/2

        K_MATRIX[1, 1, 0] := 0
        K_MATRIX[2, 2, 0] := 0
        K_MATRIX[3, 3, 0] := 1
    end

10:  pbvX := (b2+b4)/2
    pbvY := 0
    pbvZ := 0

        K_MATRIX[1, 1, 0] := 1
        K_MATRIX[2, 2, 0] := 0
        K_MATRIX[3, 3, 0] := 0
    end

12:  pbvX := 0
    pbvY := (b4-b3)/2
    pbvZ := 0

        K_MATRIX[1, 1, 0] := 0
        K_MATRIX[2, 2, 0] := 1
        K_MATRIX[3, 3, 0] := 0
    end

7:   pbvX := (b1+b3)/2
    pbvY := (b1-b2)/2
    pbvZ := (b2-b3)/2

```

```

      K_MATRIX[1, 1, 0] := 1
      K_MATRIX[2, 2, 0] := 1
      K_MATRIX[3, 3, 0] := 1
    end

11: pbvX := (b2+b4)/2
    pbvY := (b1-b2)/2
    pbvZ := (b1-b4)/2

      K_MATRIX[1, 1, 0] := 1
      K_MATRIX[2, 2, 0] := 1
      K_MATRIX[3, 3, 0] := 1
    end

13: pbvX := (b1+b3)/2
    pbvY := (b4-b3)/2
    pbvZ := (b1-b4)/2

      K_MATRIX[1, 1, 0] := 1
      K_MATRIX[2, 2, 0] := 1
      K_MATRIX[3, 3, 0] := 1
    end

14: pbvX := (b2+b4)/2
    pbvY := (b4-b3)/2
    pbvZ := (b2-b3)/2

      K_MATRIX[1, 1, 0] := 1
      K_MATRIX[2, 2, 0] := 1
      K_MATRIX[3, 3, 0] := 1
    end

15: pbvX := (b1+b2+b3+b4)/4
    pbvY := (b1-b2-b3+b4)/4
    pbvZ := (b1+b2-b3-b4)/4

      K_MATRIX[1, 1, 0] := 1
      K_MATRIX[2, 2, 0] := 1
      K_MATRIX[3, 3, 0] := 1
    end

    end

(*****
* 3D) Convert "processed" beam velocities into body velocities.
*****)

TDLR_VELOCITY[1] := pbvX / cos(TDLR_ANGLES[1])
TDLR_VELOCITY[2] := pbvY / cos(TDLR_ANGLES[2])
TDLR_VELOCITY[3] := pbvZ / cos(TDLR_ANGLES[3])

(** where cos represents the cosine function. **)

END P_SPEC

```

Example Software Code

Appendix I

* **Module:** TDLRSP.FOR

* Facility: Pluto

* P-Spec: 1.5

* Abstract:

* This module contains the implementation of the functional
* requirements for TDLRSP.

*

* List of Routines:

* subroutine TDLRSP

* Title: TDLRSP

* Facility: Pluto

* Abstract:

- * 1) Maintain the history of the vehicle velocities and the
* velocity computation indicator
- * 2) Determine the operational status of touch down landing radar
* sensor
- * 3) Report the current vehicle velocities along each of the
* vehicle's three axes
- * 4) Report the velocity computation indicators.

*

* Arguments: None

* Revision History:

* v0 15-sep-1994 Rob Angellatta (RKA) Original.

* v1 30-Nov-1994 Philip Morris (PEM)

* v2 10-JAN-1995 Philip Morris (PEM)

subroutine TDLRSP

implicit none

*** include the global common stores ***

include 'external.for'
include 'guidance_state.for'
include 'sensor_output.for'
include 'run_parameters.for'

*** include constant definitions ***

include 'constants.for'

*** declare local variables ***

integer*4 i

real*8 b(4)
real*8 pbvX
real*8 pbvY
real*8 pbvZ

real*8 elapsed_time

* 1) Maintain the history of the vehicle velocities and the
* velocity computation indicator by "rotating variables." The data

TDLR_VELOCITY(1, 4) = TDLR_VELOCITY(1, 3)
TDLR_VELOCITY(1, 3) = TDLR_VELOCITY(1, 2)
TDLR_VELOCITY(1, 2) = TDLR_VELOCITY(1, 1)
TDLR_VELOCITY(1, 1) = TDLR_VELOCITY(1, 0)

TDLR_VELOCITY(2, 4) = TDLR_VELOCITY(2, 3)
TDLR_VELOCITY(2, 3) = TDLR_VELOCITY(2, 2)
TDLR_VELOCITY(2, 2) = TDLR_VELOCITY(2, 1)
TDLR_VELOCITY(2, 1) = TDLR_VELOCITY(2, 0)

TDLR_VELOCITY(3, 4) = TDLR_VELOCITY(3, 3)
TDLR_VELOCITY(3, 3) = TDLR_VELOCITY(3, 2)
TDLR_VELOCITY(3, 2) = TDLR_VELOCITY(3, 1)
TDLR_VELOCITY(3, 1) = TDLR_VELOCITY(3, 0)

K_MATRIX(1, 1, 4) = K_MATRIX(1, 1, 3)
K_MATRIX(1, 2, 4) = K_MATRIX(1, 2, 3)
K_MATRIX(1, 3, 4) = K_MATRIX(1, 3, 3)
K_MATRIX(2, 1, 4) = K_MATRIX(2, 1, 3)
K_MATRIX(2, 2, 4) = K_MATRIX(2, 2, 3)
K_MATRIX(2, 3, 4) = K_MATRIX(2, 3, 3)
K_MATRIX(3, 1, 4) = K_MATRIX(3, 1, 3)
K_MATRIX(3, 2, 4) = K_MATRIX(3, 2, 3)
K_MATRIX(3, 3, 4) = K_MATRIX(3, 3, 3)

K_MATRIX(1, 1, 3) = K_MATRIX(1, 1, 2)
K_MATRIX(1, 2, 3) = K_MATRIX(1, 2, 2)
K_MATRIX(1, 3, 3) = K_MATRIX(1, 3, 2)
K_MATRIX(2, 1, 3) = K_MATRIX(2, 1, 2)
K_MATRIX(2, 2, 3) = K_MATRIX(2, 2, 2)
K_MATRIX(2, 3, 3) = K_MATRIX(2, 3, 2)
K_MATRIX(3, 1, 3) = K_MATRIX(3, 1, 2)
K_MATRIX(3, 2, 3) = K_MATRIX(3, 2, 2)
K_MATRIX(3, 3, 3) = K_MATRIX(3, 3, 2)

K_MATRIX(1, 1, 2) = K_MATRIX(1, 1, 1)
K_MATRIX(1, 2, 2) = K_MATRIX(1, 2, 1)
K_MATRIX(1, 3, 2) = K_MATRIX(1, 3, 1)
K_MATRIX(2, 1, 2) = K_MATRIX(2, 1, 1)
K_MATRIX(2, 2, 2) = K_MATRIX(2, 2, 1)
K_MATRIX(2, 3, 2) = K_MATRIX(2, 3, 1)
K_MATRIX(3, 1, 2) = K_MATRIX(3, 1, 1)
K_MATRIX(3, 2, 2) = K_MATRIX(3, 2, 1)
K_MATRIX(3, 3, 2) = K_MATRIX(3, 3, 1)

K_MATRIX(1, 1, 1) = K_MATRIX(1, 1, 0)
K_MATRIX(1, 2, 1) = K_MATRIX(1, 2, 0)
K_MATRIX(1, 3, 1) = K_MATRIX(1, 3, 0)
K_MATRIX(2, 1, 1) = K_MATRIX(2, 1, 0)
K_MATRIX(2, 2, 1) = K_MATRIX(2, 2, 0)

K_MATRIX(2, 3, 1) = K_MATRIX(2, 3, 0)
K_MATRIX(3, 1, 1) = K_MATRIX(3, 1, 0)
K_MATRIX(3, 2, 1) = K_MATRIX(3, 2, 0)
K_MATRIX(3, 3, 1) = K_MATRIX(3, 3, 0)

* 2) Determine the operational status of touch down landing radar sensor.

TDLR_STATUS(1) = K\$HEALTHY
TDLR_STATUS(2) = K\$HEALTHY
TDLR_STATUS(3) = K\$HEALTHY
TDLR_STATUS(4) = K\$HEALTHY

* 3) Reporting the current vehicle velocities along each of the
* vehicle's three axes and reporting the velocity computation
* indicators.

* 3A) Determine the state of the four radar beams.

*

* The data element TDLR_STATE contains the state of the radar
* beams.

*

* Valid radar beam states are "locked" (value 1) and "unlocked"
* (value 0). The present state of a radar beam is determined from
* the current value of the sensor data and the previous state of
* the radar beam. A sensor measurement of zero indicates that the
* radar beam echo was not received and the radar beam is considered
* to be "unlocked." A non-zero sensor measurement indicates that a
* radar beam echo was received, but does not imply a radar beam
* state of "locked." Because, once a radar beam is declared
* "unlocked," it is rendered unusable (remains "unlocked"
* regardless of the sensor data value) for a specified period of
* time. This waiting period must be implemented in the software.

*

* A beam is deemed "locked" when 1) the current sensor value
* contains a non-zero value and the beam's previous state was
* "locked"; or 2) the current sensor value contains a non-zero
* value and the beam's previous state was "unlocked" and the
* elapsed time since the beam was determined "unlocked" is greater
* than or equal to the sensor recovery period.

*

* The data element TDLR_LOCK_TIME specifies the unlocked sensor
* recovery (waiting) period. The data element FRAME_BEAM_UNLOCKED
* is updated with the value of the FRAME_COUNTER during the frame
* in which a radar beam state is first determined as "unlocked."
* The data element DELTA_T specifies in seconds the duration of a
* single frame. Thus the elapsed time since a radar beam was
* declared "unlocked" can be determined by subtracting the present
* value of FRAME_COUNTER from the value of FRAME_BEAM_UNLOCKED and
* multiplying the result by the value of DELTA_T.

```

**** process each radar beam ****

do 100 i=1,4

  if (TDLR_COUNTER(i) .EQ. 0) then

    if (TDLR_STATE(i) .EQ. K$BEAM_LOCKED) then
      TDLR_STATE(i) = K$BEAM_UNLOCKED
      FRAME_BEAM_UNLOCKED(i) = FRAME_COUNTER
    ****
    * v2 Changes for AR#24. Item 7. Added else if.
    ****
    *
    *   else
    *     elseif (TDLR_STATE(i) .EQ. K$BEAM_UNLOCKED) then
    ****
    * v2 Changes for AR#24. End Change.
    ****
    *
    *           the beam was unlocked
    *     elapsed_time = DELTA_T *
    *   &
    *     (FRAME_COUNTER - FRAME_BEAM_UNLOCKED(i))
    *
    *     if (elapsed_time .GE. TDLR_LOCK_TIME) then
    *       FRAME_BEAM_UNLOCKED(i) = FRAME_COUNTER
    *     end if
    *   end if
    *
    * else
    *
    *           the sensor measurement != 0
    *
    *     if (TDLR_STATE(i) .EQ. K$BEAM_UNLOCKED) then
    *       elapsed_time = DELTA_T *
    *     &
    *       (FRAME_COUNTER - FRAME_BEAM_UNLOCKED(i))
    *
    *       if (elapsed_time .GE. TDLR_LOCK_TIME) then
    *         TDLR_STATE(i) = K$BEAM_LOCKED
    *       end if
    *     end if
    *   end if
    *
    * 100 continue

    ****
    * 3B) Determine the beam velocities.
    ****

    do 200 i=1,4
      b(i) = TDLR_OFFSET + TDLR_GAIN * TDLR_COUNTER(i)
    200 continue

    ****
    * 3C) Determine the "processed" beam velocities, and
    * 4) Determine the velocity computation indicators.
    ****
    ****
    * Compute a "processed" beam velocity for each of the three axes as
    * specified by the following table:

```

```

*
* Beams | PROCESSED BEAM VELOCITIES | K-MATRIX | Case
* in lock | pbvX pbvY pbvZ | X Y Z | Number
* -----|-----|-----|-----|-----
* none | 0 | 0 | 0 | 0|0|0|0| 0
* 1 | 0 | 0 | 0 | 0|0|0|0| 1
* 2 | 0 | 0 | 0 | 0|0|0|0| 2
* 3 | 0 | 0 | 0 | 0|0|0|0| 4
* 4 | 0 | 0 | 0 | 0|0|0|0| 8
* -----|-----|-----|-----|-----
* 1,2 | 0 | (b(1)-b(2))/2 | 0 | 0|1|0| 3
* 1,3 | (b(1)+b(3))/2 | 0 | 0 | 1|0|0| 5
* 1,4 | 0 | 0 | (b(1)-b(4))/2 | 0|0|1| 9
* 2,3 | 0 | 0 | (b(2)-b(3))/2 | 0|0|1| 6
* 2,4 | (b(2)+b(4))/2 | 0 | 0 | 1|0|0| 10
* 3,4 | 0 | (b(4)-b(3))/2 | 0 | 0|1|0| 12
* -----|-----|-----|-----|-----
* 1,2,3 | (b(1)+b(3))/2 | (b(1)-b(2))/2 | (b(2)-b(3))/2 | 1|1|1| 7
* 1,2,4 | (b(2)+b(4))/2 | (b(1)-b(2))/2 | (b(1)-b(4))/2 | 1|1|1| 11
* 1,3,4 | (b(1)+b(3))/2 | (b(4)-b(3))/2 | (b(1)-b(4))/2 | 1|1|1| 13
* 2,3,4 | (b(2)+b(4))/2 | (b(4)-b(3))/2 | (b(2)-b(3))/2 | 1|1|1| 14
* -----|-----|-----|-----|-----
* 1,2,3,4 | a | b | c | 1|1|1| 15
*
* a) (b(1)+b(2)+b(3)+b(4))/4
* b) (b(1)-b(2)-b(3)+b(4))/4
* c) (b(1)+b(2)-b(3)-b(4))/4
*
* Each of the 16 possible cases has been assigned a case number to
* facilitate the description of the necessary processing. The case
* number is found in the column labeled "Case Number" in the table
* above.
*
* Determine the case number value for the current processing.
* Each of the four radar beams' state has been assigned a weight
* value: beam 1: 1, beam 2: 2, beam 3: 4, beam 4: 8. The "case
* number" is computed by summing the radar beams multiplied by their
* their weight factors.
*****

***
* v1 Changes for AR#23. Item 24. Default goto 2000 added.
***
go to (1000,1000,1000,1010,1000,1020,1040,1070,
& 1000,1030,1050,1080,1060,1090,1100,1110),
& TDLR_STATE(1) + 2*TDLR_STATE(2) +
& 4*TDLR_STATE(3) + 8*TDLR_STATE(4) + 1
go to 2000
***
* v1 Changes for AR#23. End Change.
***
*** cases 0, 1, 2, 4, 8 ***

1000 pbvX = 0.0
pbvY = 0.0
pbvZ = 0.0

```

```
K_MATRIX(1, 1, 0) = 0
K_MATRIX(2, 2, 0) = 0
K_MATRIX(3, 3, 0) = 0
go to 2000
```

*** case 3 ***

```
1010  pbvX = 0.0
      pbvY = (b(1) - b(2)) / 2.0
      pbvZ = 0.0
```

```
K_MATRIX(1, 1, 0) = 0
K_MATRIX(2, 2, 0) = 1
K_MATRIX(3, 3, 0) = 0
go to 2000
```

*** case 5 ***

```
1020  pbvX = (b(1) + b(3)) / 2.0
      pbvY = 0.0
      pbvZ = 0.0
```

```
K_MATRIX(1, 1, 0) = 1
K_MATRIX(2, 2, 0) = 0
K_MATRIX(3, 3, 0) = 0
go to 2000
```

*** case 9 ***

```
1030  pbvX = 0.0
      pbvY = 0.0
      pbvZ = (b(1) - b(4)) / 2.0
```

```
K_MATRIX(1, 1, 0) = 0
K_MATRIX(2, 2, 0) = 0
K_MATRIX(3, 3, 0) = 1
go to 2000
```

*** case 6 ***

* v1 Changes for AR#23. Item 25. Goto 2000 added to finish the case properly

```
1040  pbvX = 0.0
      pbvY = 0.0
      pbvZ = (b(2) - b(3)) / 2.0
```

```
K_MATRIX(1, 1, 0) = 0
K_MATRIX(2, 2, 0) = 0
K_MATRIX(3, 3, 0) = 1
go to 2000
```

* v1 Changes for AR#23. End Change.

*** case 10 ***

```
1050   pbvX = (b(2) + b(4)) / 2.0
      pbvY = 0.0
      pbvZ = 0.0

      K_MATRIX(1, 1, 0) = 1
      K_MATRIX(2, 2, 0) = 0
      K_MATRIX(3, 3, 0) = 0
      go to 2000
```

*** case 12 ***

```
1060   pbvX = 0.0
      pbvY = (b(4) - b(3)) / 2.0
      pbvZ = 0.0

      K_MATRIX(1, 1, 0) = 0
      K_MATRIX(2, 2, 0) = 1
      K_MATRIX(3, 3, 0) = 0
      go to 2000
```

*** case 7 ***

```
1070   pbvX = (b(1) + b(3)) / 2.0
      pbvY = (b(1) - b(2)) / 2.0
      pbvZ = (b(2) - b(3)) / 2.0

      K_MATRIX(1, 1, 0) = 1
      K_MATRIX(2, 2, 0) = 1
      K_MATRIX(3, 3, 0) = 1
      go to 2000
```

*** case 11 ***

```
1080   pbvX = (b(2) + b(4)) / 2.0
      pbvY = (b(1) - b(2)) / 2.0
      pbvZ = (b(1) - b(4)) / 2.0

      K_MATRIX(1, 1, 0) = 1
      K_MATRIX(2, 2, 0) = 1
      K_MATRIX(3, 3, 0) = 1
      go to 2000
```

*** case 13 ***

```
1090   pbvX = (b(1) + b(3)) / 2.0
      pbvY = (b(4) - b(3)) / 2.0
      pbvZ = (b(1) - b(4)) / 2.0

      K_MATRIX(1, 1, 0) = 1
      K_MATRIX(2, 2, 0) = 1
      K_MATRIX(3, 3, 0) = 1
      go to 2000
```

*** case 14 ***

```
1100   pbvX = (b(2) + b(4)) / 2.0
      pbvY = (b(4) - b(3)) / 2.0
      pbvZ = (b(2) - b(3)) / 2.0
```

```
      K_MATRIX(1, 1, 0) = 1
      K_MATRIX(2, 2, 0) = 1
      K_MATRIX(3, 3, 0) = 1
```

```
      go to 2000
```

```
*** case 15 ***
```

```
1110   pbvX = (b(1) + b(2) + b(3) + b(4)) / 4.0
      pbvY = (b(1) - b(2) - b(3) + b(4)) / 4.0
      pbvZ = (b(1) + b(2) - b(3) - b(4)) / 4.0
```

```
      K_MATRIX(1, 1, 0) = 1
      K_MATRIX(2, 2, 0) = 1
      K_MATRIX(3, 3, 0) = 1
```

```
2000 continue
```

```
*****
```

```
* 3D) Convert "processed" beam velocities into body velocities.
```

```
*****
```

```
      TDLR_VELOCITY(1, 0) = pbvX / COS(TDLR_ANGLES(1))
      TDLR_VELOCITY(2, 0) = pbvY / COS(TDLR_ANGLES(2))
      TDLR_VELOCITY(3, 0) = pbvZ / COS(TDLR_ANGLES(3))
```

```
      return
      end
```

```
***** end of module tdlrsp.for *****
```

Example Software Test Data

Appendix J

Example Test Case for TDLRSP

3.6 TDLRSP Functional Unit Test Cases

Table 8 is a listing of all test cases for the TDLRSP functional unit. All test cases manipulate the variables:

FRAME_COUNTER	TDLR_COUNTER
FRAME_BEAM_UNLOCKED	TDLR_STATE
K_MATRIX	TDLR_VELOCITY

For robustness testing purposes, Table 5.11 of the GCS Specification is missing several cases that should be tested. These conditions are given in Table 7 below. Note that the *Beam_lock_time* calculated by:

$$Beam_lock_time = DELTA_T * (FRAME_COUNTER - FRAME_BEAM_UNLOCKED)$$

Table 7 also identifies the test cases for each of those conditions. These cases are also repeated in Table 8.

Table 7: Conditions not given in Table 5.11 of the GCS Specification.

Input			Output		Test Case
TDLR_STATE	TDLR_COUNTER	<i>Beam_lock_time</i> ≥ TDLR_LOCK_TIME	TDLR_STATE	FRAME_BEAM_UNLOCKED	Names
locked	≠ 0	d	locked	Unchanged	TDLRSP_RO_006.TC
unlocked	≠ 0	no	unlocked	Unchanged	TDLRSP_RO_002.TC
unlocked	= 0	no	unlocked	Unchanged	TDLRSP_RO_004.TC

Table 8: Test cases for TDLRSP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
tdlrsp_nr_001.m	Test: 1) TDLR_STATE = 0 & TDLR_COUNTER != 0 (line 2 of table 5.11) 2) line 16 of table 5.12 2) history rotation for TDLR_VELOCITY & K_MATRIX	tdlrsp_nr_001.tc	tdlrsp_nr_001.ex
tdlrsp_ro_002.m	Test: 1) TDLR_STATE = 0 & TDLR_COUNTER != 0 but elapsed time < TDLR_LOCK_TIME (not listed in table 5.11)	tdlrsp_ro_002.tc	tdlrsp_ro_002.ex
tdlrsp_nr_003.m	Test: TDLR_STATE = 0 & TDLR_COUNTER = 0 (line 3 of table 5.11)	tdlrsp_nr_003.tc	tdlrsp_nr_003.ex
tdlrsp_ro_004.m	Test: TDLR_STATE = 0 & TDLR_COUNTER = 0 but elapsed time < TDLR_LOCK_TIME (not listed in table 5.11)	tdlrsp_ro_004.tc	tdlrsp_ro_004.ex
tdlrsp_nr_005.m	Test: 1) TDLR_STATE = 1 & TDLR_COUNTER = 0 (line 1 of table 5.11) 2) line 1 of table 5.12 (no beams in lock)	tdlrsp_nr_005.tc	tdlrsp_nr_005.ex
tdlrsp_ro_006.m	Test: 1) TDLR_STATE = 1 & TDLR_COUNTER != 0 (not listed in table 5.11) 2) line 1 of table 5.12 (no beams in lock)	tdlrsp_ro_006.tc	tdlrsp_ro_006.ex
tdlrsp_nr_007.m	Test: Beam 1 in lock (line 2 of table 5.12)	tdlrsp_nr_007.tc	tdlrsp_nr_007.ex
tdlrsp_nr_008.m	Test: Beam 2 in lock (line 3 of table 5.12)	tdlrsp_nr_008.tc	tdlrsp_nr_008.ex
tdlrsp_nr_009.m	Test: Beam 3 in lock (line 4 of table 5.12)	tdlrsp_nr_009.tc	tdlrsp_nr_009.ex
tdlrsp_nr_010.m	Test: Beam 4 in lock (line 5 of table 5.12)	tdlrsp_nr_010.tc	tdlrsp_nr_010.ex
tdlrsp_nr_011.m	Test: Beams 1 & 2 in lock (line 6 of table 5.12)	tdlrsp_nr_011.tc	tdlrsp_nr_011.ex
tdlrsp_nr_012.m	Test: Beams 1 & 3 in lock (line 7 of table 5.12)	tdlrsp_nr_012.tc	tdlrsp_nr_012.ex
tdlrsp_nr_013.m	Test: Beams 1 & 4 in lock (line 8 of table 5.12)	tdlrsp_nr_013.tc	tdlrsp_nr_013.ex
tdlrsp_nr_014.m	Test: Beams 2 & 3 in lock (line 9 of table 5.12)	tdlrsp_nr_014.tc	tdlrsp_nr_014.ex
tdlrsp_nr_015.m	Test: Beams 2 & 4 in lock (line 10 of table 5.12)	tdlrsp_nr_015.tc	tdlrsp_nr_015.ex
tdlrsp_nr_016.m	Test: Beams 3 & 4 in lock (line 11 of table 5.12)	tdlrsp_nr_016.tc	tdlrsp_nr_016.ex
tdlrsp_nr_017.m	Test: Beams 1, 2, & 3 in lock (line 12 of table 5.12)	tdlrsp_nr_017.tc	tdlrsp_nr_017.ex
tdlrsp_nr_018.m	Test: Beams 1, 2, & 4 in lock (line 13 of table 5.12)	tdlrsp_nr_018.tc	tdlrsp_nr_018.ex
tdlrsp_nr_019.m	Test: Beams 1, 3, & 4 in lock (line 14 of table 5.12)	tdlrsp_nr_019.tc	tdlrsp_nr_019.ex
tdlrsp_nr_020.m	Test: Beams 2, 3, & 4 in lock (line 15 of table 5.12)	tdlrsp_nr_020.tc	tdlrsp_nr_020.ex
tdlrsp_nr_021.m	Test: ALL Beams in lock (line 16 of table 5.12)	tdlrsp_nr_021.tc	tdlrsp_nr_021.ex
tdlrsp_ro_022.m	Test FRAME_BEAM_UNLOCKED out of UPPER bound	tdlrsp_ro_022.tc	tdlrsp_ro_022.ex
tdlrsp_ro_023.m	Test FRAME_BEAM_UNLOCKED out of LOWER bound	tdlrsp_ro_023.tc	tdlrsp_ro_023.ex
tdlrsp_ro_024.m	Test FRAME_COUNTER out of UPPER bound	tdlrsp_ro_024.tc	tdlrsp_ro_024.ex
tdlrsp_ro_025.m	Test FRAME_COUNTER out of LOWER bound	tdlrsp_ro_025.tc	tdlrsp_ro_025.ex
tdlrsp_ro_026.m	Test TDLR_STATE INVALID value	tdlrsp_ro_026.tc	tdlrsp_ro_026.ex
tdlrsp_ro_027.m	Test TDLR_COUNTER out of LOWER bound	tdlrsp_ro_027.tc	tdlrsp_ro_027.ex
tdlrsp_ro_028.m	Test TDLR_COUNTER out of UPPER bound	tdlrsp_ro_028.tc	tdlrsp_ro_028.ex

Example Test Results for TDLRSP

3.1.6 TDLRSP Functional Unit

Code components tested for TDLRSP are given in Table 12.

Table 12: TDLRSP code components.

EXTERNAL.FOR	TDLRSP.FOR
RUN_PARAMETERS.FOR	UTILITY.FOR
GUIDANCE_STATE.FOR	CONSTANTS.FOR
SENSOR_OUTPUT.FOR	

Total number of normal range (NR) test cases: 18

Total number of robustness (RO) test cases: 10

The ANA file generated for TDLRSP_RO_026 involves a condition that is not specified in the SPEC. Although the results of this test run does not agree with the expected values, the results are just as valid because this robustness test case exercises a condition that is not defined in the Specification. More specifically, a value of "2" is assigned to the variable TDLR_STATE. Although a "2" is not defined as a legal value for this variable in the GCS Spec, it is a possible value since the variable is ultimately implemented as an integer. For robustness test cases, DO-178B requires only that the software not cause any detrimental effects to the system. For this specific test case, the PLUTO code leaves the values of K_MATRIX unchanged. This will not have a severe impact on the implementation's ability to deliver the required function for TDLRSP.

Table 13: Summary of Requirements-based Testing on the TDLRSP Functional Unit.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
TDLRSP_NR_XXX	1/4/95	N	Initial testing
TDLRSP_RO_XXX		N	
TDLRSP_RO_026		Y/24	
TDLRSP_NR_XX	1/13/95	N	Retesting due to PR 24.
TDLRSP_RO_XXX		N	
TDLRSP_RO_026		Y	
TDLRSP_NR_XX	4/7/95	N	Retest after Cases & Procedures finalized.
TDLRSP_RO_XXX		N	
TDLRSP_RO_026		Y	

Software Needs Assessment

Appendix K

Software Needs Assessment for Engineers

Introduction to the Survey

A. Purpose

The Software Needs Assessment will collect **aggregate** data from engineers who currently approve software, who will approve software, or who develop guidance for software. Individual respondents will be anonymous. The results will be used to determine how best to support engineers in their performance of Aircraft Certification's software functions. The organizational profiles obtained from this instrument will be used to develop training and to evaluate staffing initiatives.

No individual is currently expected to have all the relevant software knowledge and skills. Varying skill levels exist within the Aircraft Certification organizations. Engineers have done a good job with their software responsibilities, considering that FAA-offered training has been very limited.

The following types of data will be collected in the Survey:

- . ACO/Directorate/Division software workload
- . Type/amount of each organization's current software knowledge, skill and experience in relationship to the required knowledge, skill and ability

Data will be used to do the following:

- . Identify level of software expertise in each organization
- . Describe skill gaps, if existing, in each organization
- . Identify type and level of training and development activities required
- . Provide input for staff planning and development

B. Knowledge, Skills and Abilities

Each question in the survey measures a required software knowledge, skill or ability (KSA). These KSAs were obtained from the following sources: (1) Software Grand Design Report, (2) Order 8110.37B (Designated Engineering Representative System), and (3) the Airborne Software Substantiation Course Design Guide. These KSAs are considered to be the minimum software expertise needed to evaluate software by applying DO-178B.

Software Needs Assessment for Engineers

The KSAs include:

1. Knowledge of the system safety assessment process in order to establish the software criticality level.
2. Knowledge of the rationale for, and the significance of, each stage in the software development process, as well as its supporting standards, procedures and documentation.
3. Ability to apply knowledge of all phases of the software development life cycle addressed in DO-178B, including the testing processes and configuration and quality control procedures.
4. Sufficient knowledge of at least one high level and one assembly level programming language, as well as, knowledge of typical support software used in the software development process in order to be able to evaluate potential problems with the coding and execution process.
5. Knowledge of sources of software anomalies (e.g. errors), relative merits of types of testing procedures, and characteristics of a thorough test program.
6. Knowledge of computing as it relates to a real-time avionics system, e.g., use of interrupts and multitasking.
7. Knowledge of hardware characteristics and their impact on software interface and potential to generate anomalies.

C. Who Will Take This Survey?

All engineers who currently approve or will approve software data submitted under DO-178B or have responsibility for developing guidance for evaluation of software.

D. Survey Contents

This is a **self** assessment instrument with two sets of questions: (1) current software experience and (2) specific software knowledge, skill, and ability.

Section A of the Survey assesses the software workload activity in the organization.

Section B addresses the degree of software knowledge relative to the KSAs listed above. The format for this Section includes: (1) the KSA that is being measured, (2) examples of situations that might arise if the required KSA is **not** evident, i.e. anomalies and (3) questions that measure that KSA.

Software Needs Assessment for Engineers

II. SOFTWARE NEEDS ASSESSMENT INSTRUMENT

NOTE: This Assessment is not intended to evaluate individuals. Only aggregate data will be collected.

****Please Provide Answers in the Space Provided and on the Attached Answer Sheet ****

Section A. Current Software Experience

1. Are you currently doing software approvals ? ___yes ___no
 2. Rate your relative comfort level if asked to perform a software evaluation on your own? _____
(0) low (1) moderate (2) high
 3. **A.** What **number** of software approvals (including DER submittal approvals and software approval letters) have you been **involved with** in your organization for systems, components, etc. over the past year (e.g. parts of TCs, STCs, TSOs, changes, etc.)? (If you haven't been employed for a year, project what you would have done based on your assignments to date). # _____
 - B.** What is the **approximate percent** of approvals **made** in the following categories? (0-100%)
 1. TSO projects with software _____%
 2. TC /ATC systems with software _____%
 3. STC systems with software _____%
 4. TC or STC with TSO authorized equipment) _____%
 - C.** What is the **approximate percent** of time devoted to software approval?
_____%
 - D.** Not Applicable (NA) (i.e. software approval is not part of your job _____)
4. What is the **approximate number** of software DERs that you supervise.

Software Needs Assessment for Engineers

Section A. Current Software Experience (Cont')

5. Provide the figures below that represent the **software review workload** you have been involved with in the past year. (If you haven't been employed for a year, project what you would have done based on your assignments to date). **Write NA if any question is Not Applicable.**

Approximate number of reviews made in the following categories:

(A) on-site reviews # _____

(B) desk review # _____

(C) DER delegation # _____

(D) Other # _____

6. Provide the **approximate number** of policy/guidance projects (e.g. committees, issue papers, review and creation of orders, ACs, regulations, etc.) you have been involved with regarding the production or review of software related issues within the last year. (If you haven't been employed for a year, project what you would have done based on your assignments to date)

7. Approximately what **percent** change do you see in the number of software approvals for which you will be responsible. _____%

8. What percent of on-site software reviews that you conduct are **manufacturing inspectors invited** to team with you? _____%

9. How often do the **invited** inspectors accept the offer to participate in software reviews?

1 - almost always

2 - most of the time

3 - sometimes

4- rarely

10. **Office identifier** (e.g. ANM-100S) _____

Software Needs Assessment for Engineers

Section B. Specific Software Knowledge, Skill, and Ability

Scoring for Knowledge, Skill, and Ability Questions

Read each question and provide a rating of 1 to 4. Your rating indicates to what degree you feel capable of answering the questions.

Answers to the questions are given to minimize the confusion as to what constitutes a comprehensive response. You can therefore better judge to what degree you possess the knowledge to answer the question. For example, if you can define terms, but can't answer any part of the question, rate yourself a "2". If you are able to answer the complete question, rate yourself a "4". Your rating should reflect your knowledge of the subject matter. It is included to help clarify the question's meaning or serve to jog your memory. Rate yourself as follows:

- One (1) - No experience (i.e., not able to answer the question)
- Two (2) - Some or little (minimal familiarity with content area)
- Three (3) - Moderate (able to answer some part of question, require resources to answer completely)
- Four (4) - Considerable (comprehensive understanding, know or have studied principles associated with content)

There are 7 KSAs and a total of 24 questions that measure them.

Software Needs Assessment for Engineers

KSA #1 Knowledge of the system safety assessment process in order to establish the software criticality levels.

➤ Examples to justify why KSA is needed -

The applicant has chosen the wrong software level.

The applicant hasn't considered the effect of other software components.

QUESTIONS For KSA #1: Read each question and provide a rating of 1 to 4. Your rating indicates to what degree you feel capable of answering the questions.

1.1 Can you draw a fault tree and show how software fits in? Is software handled as a 1 or a zero? (Rate 1-4) _____

This is somewhat of a difficult question. Basically, software is included in a fault tree only to see how it contributes to a given failure condition and whether there are any mitigating circumstances. When the fault tree is used to calculate probabilities, the software branch can be removed. Another approach is to give software a value of "1" for all AND gates and a value of "0" for all OR gates which has the same effect as removal.

1.2 Can you explain the relationship between ARP 4754 and DO-178B in establishing software level? (Rate 1-4) _____

- Although ARP 4754 hasn't been officially recognized, it provides the basis for establishing software levels. The material in DO-178B section 2 was intended to be informative rather than normative with the expectation that all questions regarding software levels would be addressed in ARP 4754.

1.3 Consider a cruise autopilot where the software developed to level D has the capability of producing a pitch hard-over which could unacceptably overstress the aircraft (catastrophic failure condition) and a monitor in software which could disconnect the autopilot prior to any damage. What is the software level of the monitor and some important considerations? (Rate 1-4) _____

According to DO-178B and ARP 4754, the monitor software has to be assured to level A and be protected (e.g. partitioned) from the other software. However since ARP 4754 is now the informal governing guidance the lowest level of the cruise autopilot needs to be at level C for this situation. Also, according to ARP 4754 the hardware reliability for the hardware delivering the monitor effects must be equal to or greater than the main command path.

Software Needs Assessment for Engineers

KSA #2 Knowledge of the rationale for, and the significance of, each stage in the software development process, as well as its supporting standards, procedures, and documentation.

➤ **Examples to justify why KSA is needed:**

An applicant claims data coverage because they are using a data flow methodology tool.

An applicant provides a functional test that doesn't provide the degree of structural coverage claimed.

An applicant's plans state they are doing a waterfall approach, but their data indicate incremental development which means the process is not controlled by the plans. The CM and QA probably don't match as well.

An applicant claims that a McCabe complexity metric of 45 provides adequate code review criteria.

An applicant claims a tool provides adequate test coverage, but it doesn't

An applicant claims that their tool provides configuration management, but it doesn't.

Questions for KSA #2: Read each question and provide a rating of 1 to 4. Your rating indicates to what degree you feel capable of answering the questions.

2.1 Rate the degree of your expertise in each of the following areas.: (Rate 1-4)

- A. S/W Requirements creation** _____
- B. S/W Design methodologies** _____
- C. S/W Coding** _____
- D. S/W Verification** _____
- E. Reviews/Walkthroughs** _____
- F. Testing** _____
- G. S/W Configuration Management** _____
- H. S/W Quality Assurance** _____
- I. S/W Metrics** _____

Software Needs Assessment for Engineers

2.2 *What is the relationship of where an error is introduced in a software development lifecycle and where it is found?*

(Rate 1-4) _____

- The larger the distance between discovery and correction the larger the costs. Typically this is a non-linear relationship. Industry has published that it is 10-100 times more costly to fix a software problem discovered in service than if it was discovered and fixed during the requirements phase

2.3 *What is the relationship between number and type of problems which could be found doing only software component testing Vs the problems which would be found doing only code reviews on the same software component? (Rate 1-4) _____*

- Module testing and code reviews overlap almost 90% in that they will detect similar type of errors. Clearly, compiler and linker errors will not get caught nor will a number of run time errors be caught by code reviews.

2.4 *What are some definitions of a software baseline?*

(Rate 1-4) _____

- Although there are formal definitions of baselines, a more practical definition would be where one organization or person needs to communicate stable configuration to a different organization or person. It is then that the collection of items (e.g. documentation, code, design) is identified and put under some form of defined control (e.g. configuration management). The control is designed such that unauthorized changes will be prevented/detected and authorized changes will be communicated in a timely manner to affected parties. This controlled collection will define a baseline. (This is similar to the glossary definition in DO-178B)

2.5 *What is your knowledge of at least one design methodology. (e.g. data flow – Ward-Mellor, Ganes & Sarson, State Machine – state mate, object oriented development - Booch, HIPO, etc.*

(Rate 1-4) _____

2.6 *What are some testing techniques? (Rate 1-4) _____*

- Partitioning testing
- Data flow testing
- Logic testing
- Mutation testing
- Random testing

2.7 *What are the problems with applying metrics (e.g. McCabes complexity, Halstead, lines of code, etc.) to software.*

(Rate 1-4) _____

- No accepted level of goodness
- Problems in collecting the data.
- Selection of a metric that is representative of the desired goal

Software Needs Assessment for Engineers

KSA #3 Ability to apply knowledge of all phases of the software development life cycle addressed in DO-178B, including the testing processes and configuration and quality control procedures.

➤ **Examples to justify why KSA is needed:**

An applicant claims that the QA records are what the industry is doing.

An applicant claims that no design is needed and very few people have anything other than requirements and heavily commented code.

An applicant's software development plan is not doable particularly in areas affecting DO-178B objectives

Questions for KSA #3: Read each question and provide a rating of 1 to 4. Your rating indicates to what degree you feel capable of answering the questions.

3.1 Rate your involvement as a production team member in each of the following lifecycle phases . This could be covered by active participation, a verifier or active quality assurance person.

(Rate each 1-4 or NA)

- A. SW Requirements creation** _____
- B. SW Design methodologies** _____
- C. SW Coding** _____
- D. SW Verification** _____
- E. Reviews/Walkthroughs** _____
- F. Testing** _____
- G. SW Configuration Management** _____
- H. SW Quality Assurance** _____
- I. SW Metrics** _____

3.2 What are the issues surrounding an approach that purports to go from software requirements (high level requirements) direct to code? (Rate 1-4) _____

- In some cases a large number of software requirements inherited from the system requirements are actually at a level of detail that they can be coded directly. However, the software architecture and other features of the design need to be documented to provide an appropriate level of abstraction. Although the practice of coding and then reverse engineering the design is not unacceptable, it should alert the certification engineer to examine the traceability and the design to see if the gap between software requirements and code is adequately bridged. The adequacy of the bridge would be demonstrated if a hypothetical new engineer can understand the relationship between code and software requirements.

Software Needs Assessment for Engineers

3.3 *What are some of the pitfalls of a rapid prototyping environment? (Rate 1-4) _____*

- There may be no defined traceability between the actual code and the software requirements.
- The code may not be robust against the requirements
- Configuration control could be lost.
- The design may not provide an appropriate level of abstraction to allow complete evaluation of the testing and verification.
- However with appropriate controls and planning these obstacles can be overcome

Software Needs Assessment for Engineers

KSA #4 Sufficient knowledge of at least one high level and one assembly level programming language. Knowledge of typical support software used in the software development process in order to be able to evaluate potential problems with the coding and execution process.

➤ **Examples to justify why KSA is needed:**

An applicant claims a group of tests provide test coverage for a given code segment, but it doesn't

An applicant has made incorrect assertions about source code to object code correspondence.

An applicant asserts that they can do their testing on a host with a same language but different processor and then recompile to the target without further testing.

Questions for KSA #4: Read each question and provide a rating of "1 to 4". Your rating indicates to what degree you feel capable of answering the questions

4.1 To what degree are you able to write a program in one high level language (FORTRAN, Ada, Pascal, PLm, Algol, C, C++, etc.) compile, link, and debug it? (Rate 1-4) _____

4.2 To what degree can you write a program in an assembly language (68000, 80X86, 8057, Z80, etc.) and assemble, link, and debug it? (Rate 1-4) _____

Software Needs Assessment for Engineers

KSA #5 Knowledge of the sources of software anomalies, the relative merits of the types of testing procedures which are available to protect against them, and the characteristics of a thorough test program.

➤ **Examples to justify why KSA is needed:**

Problem reports (all closed) could reveal life cycle issues.

The applicant does not have testing that could reveal deadlock problems in multitasking kernels.

The applicant's structural coverage analysis is not complete.

An applicant asserts that all structural coverage testing can be done on a VAX computer for a Level A autopilot using an Intel 80486.

Questions for KSA #5: Read each question and provide a rating of “1 to 4 “. Your rating indicates to what degree you feel capable of answering the questions.

5.1 *What types of problems could be generated during the linking process and what types of testing could assure that these are minimized? (Rate 1-4) _____*

- Different variables could be assigned to the same address (data flow testing,)
- Change to variables in different modules that have the same name and therefore should be at the same address are not .

5.2 *What can be deduced from an extensive number of problems found during structural testing? (Rate 1-4) _____*

- There are some major problems with previous life cycle phases. Additional analysis would be needed, but almost everyone agrees that this is a good indicator of lack of good previous phase development and possibly verification and to a lesser extent bad design. This also indicates there is a significant probability of additional errors to be found during in-service.

5.3 *What type of error would be detected using decision coverage, but not detected during statement coverage? (Rate 1-4) _____*

- In the case of the empty “else” (e.g. IF A then S1 S2 S3 ENDIF) statement coverage would fail to determine if the program operated correctly if A was false.)

Software Needs Assessment for Engineers

KSA #6 Knowledge of computing as it relates to a real-time avionics system, e.g., use of interrupts, multitasking.

➤ **Examples to justify why KSA is needed:**

A basic Rate Monatomic Analysis (RMA) is presented showing that the system is able to be scheduled, but some of the assumptions for the basic RMA model have been violated

There is no analysis showing freedom from deadlock, livelock and other real time issues

The timing analysis for a Round Robin Scheduler uses a monitor which tracked free time available. This monitor was active from the first unit tested all the way through flight test. The potentially incorrect assertion is that there is 34.6% timing margin available worst case.

Questions for KSA #6: Read each question and provide a rating of “1 to 4”. Your rating indicates to what degree you feel capable of answering the questions.

6.1 How is timing analysis done for a round robin scheduler? What would you expect from an applicant who provided timing analysis? (Rate 1-4) _____

For a round robin scheduler, the applicant should have an analysis showing worst case timing paths through the program. The applicant can either provide a computed timing analysis based on the instruction execution time from the processor user or technical manual or actually provide a test condition at the worst case timing scenario and record the timing margin. In most cases, the worst case path, although feasible from the program logic, is unfeasible from the physical constraint of the operating environment. This can be accepted providing the analysis demonstrating this is documented. Also the use of caching and any branch prediction techniques needs to be considered as part of the analysis

6.2 How is timing analysis done for a multitasking operating system? What would you expect from an applicant provided timing analysis? (Rate 1-4) _____

- Rate Monotonic Analysis (RMA) is the basic approach accepted for measuring the capability of a multitasking system for meeting hard real time deadlines. However, there are a number of sub-models associated with various assumptions in how the multitasking system is implemented. These assumptions need to be validated for a particular combination of operating systems and RMA model.

Software Needs Assessment for Engineers

KSA #7 Knowledge of hardware characteristics which have an impact on the software interface and the potential for the creation of anomalies
--

➤ **Examples to justify why KSA is needed:**

An applicant asserts their system enforces partitioning between programs with different software level assurances, but there are memory overlaps between the two programs.

An applicant asserts that they can do their testing on a host with a same language, but different processor and then recompile to the target without further testing.

Questions for KSA #7: Read each question and provide a rating of “1 to 4”). Your rating indicates to what degree you feel capable of answering the questions.

7.1 How does in interrupt controller work? (Rate 1-4) _____

- An interrupt controller is typically used to expand the number of interrupts a processor can handle. A typical controller can be programmed by the processor to set up priorities, masking, etc.

7.2 How are memory protection zones set up? (Rate 1-4) _____

Not all computers have the capability to have built-in memory protection. The ones that do typically have two or more different processor modes. One mode is privileged and has access to all instructions of the computer. The other mode has a restricted set of instructions. In the privileged mode, the computer can set up various registers or other hardware elements that allow program segments to be associated with specific memory areas for both instructions and data and have the hardware enforce memory references to only the areas allowed. Accesses outside the approved areas usually result in an exception/interrupt to the processor while preventing access to the illegal memory. These protections can be set for read, write, or both. If the processors do not have built-in memory protection, it can be produced by adding extra components to the computer.

7.3 How does instruction/memory cache affect timing analysis and what can be done to overcome these problems? (Rate 1-4) _____

- Instruction/memory cache can markedly change (usually increase) the apparent speed of a processor. When timing measurements are made under round robin scheduler, and the timing is near the 0% margin, extra effort and analysis is required to ensure a useable margin or ensure that architectural issues can handle frame overruns. Similar concerns need to be addressed when using a rate montonic analysis.

Software Needs Assessment for Engineers

7.4 *What is an In circuit emulator and how might the output be used? (Rate 1-4) _____*

An in circuit emulator is a device to replace the main processor of a computer system so that more visibility into the micro operation of the program and hardware can be examined. This allows transparent (or almost transparent) ability to monitor detailed operation of an executing program by setting breakpoints based on data or branching criteria as well as providing a number of real time representations of processor operation. In some cases the outputs from this may be needed to meet the expected results of system and module test procedures.

PLEASE PROVIDE ANY COMMENTS ON THE NEEDS ASSESSMENT ON THE BACK OF THIS PAGE, IF NEEDED.

Software Needs Assessment for Engineers

Course Evaluation Form

Appendix L

Course Evaluation Form

Please give us your candid opinions concerning the training you've just completed. Your evaluation of the IVT course is important to us and will help us provide the best possible products and service to you.

Course title: **Understanding AIR's Software Approval Process – A Course for Managers**

Date: **November 28, 2001**

Number of years of FAA experience: _____

(Optional)

Name:

Office phone: ()

For the following, please completely darken the circle appropriate to your response.

	Very Good A	Good B	Average C	Poor D	Very Poor E	N/A F
1. Length of course	<input type="radio"/>					
2. Depth of information	<input type="radio"/>					
3. Pace of training	<input type="radio"/>					
4. Clarity of objects	<input type="radio"/>					
5. Sequence of content	<input type="radio"/>					
6. Amount of activities/practice	<input type="radio"/>					
7. Quality of course materials	<input type="radio"/>					
8. Effectiveness of instructor	<input type="radio"/>					
9. Overall quality of the course	<input type="radio"/>					
10. Overall effectiveness of the IVT forms	<input type="radio"/>					

Please send this completed form to your Directorate/Division Training Manager (ATM). Thank you.

NOTES:
