BY ORDER OF THE
SECRETARY OF THE AIR FORCE

AIR FORCE MANUAL 91-119

1 FEBRUARY 1999

Safety

SAFETY DESIGN AND EVALUATION CRITERIA FOR NUCLEAR WEAPON SYSTEMS SOFTWARE

OPR: HQ AFSC/SEWE (Maj Mark Burton)

Certified by: HQ USAF/SE
(Maj Gen Francis C. Gideon, Jr.)
Pages: 34
Distribution: F

This manual contains the minimum design and evaluation criteria for software requiring nuclear safety certification. It applies to all organizations that design, develop, modify, evaluate, or operate a nuclear weapon system. It does not apply to Air Force Reserve and Air National Guard. Send recommendations for improvements to HQ AFSC/SEW, 9700 G Avenue, Kirtland AFB NM 87117-5670.

Chapter 1—GENERAL STANDARDS AND CONTROL **4**

1.1. Terms and Definitions. 4
1.2. History and Philosophy. 4
1.3. Department of Defense (DoD) SafetyStandards. 4
1.4. Development Standards. 4
1.5. NSA Certification. 5
1.6. DOE Certification. 5
1.7. Air Force Criteria. 5
1.8. Nuclear Safety Software Categories. 5
1.9. Development Process. 6
1.10. Requests for Deviations. 6
1.11. Software Advisory Group (SAG). 6
1.12. Records Disposition. 6

Chapter 2—DESIGN CRITERIA FOR NUCLEAR WEAPON SYSTEMS SOFTWARE **7**

2.1. Application. 7

2.2. Software Specifications. 7
2.3. Higher-Order Language (HOL). 7
2.4. Hierarchical Design. 7
2.5. Fault Detection. 7
2.6. Fault Response. 8
2.7. Real-Time Software. 8
2.8. Responsiveness. 8
2.9. Instruction Alterations. 8
2.10. Initialization and Shutdown. 8
2.11. Memory Characteristics. 8
2.12. Validity Checks. 8
2.13. Operating System (OS) and Run-Time-Executive (RTE). 9
2.14. Operator Interface. 9
2.15. Traceability. 9
2.16. Critical Function Call and Entry Checks. 9
2.17. Command and Data Word Format. 9
2.18. Software Configuration. 9
2.19. Hardware/software Interactions. 10
2.20. Unique Aircraft Requirements. 10
2.21. Unique ICBM Requirements. 10

Chapter 3—EVALUATION CRITERIA FOR NUCLEAR WEAPON SYSTEMS SOFTWARE 11

3.1. Evaluation Program. 11
3.2. Software Standards Compliance. 11
3.3. Critical Data Element Analysis. 11
3.4. Hierarchical Design. 11
3.5. Fault Detection. 11
3.6. Fault Response. 11
3.7. Real-Time Software. 11
3.8. Responsiveness. 12
3.9. Alterations Detection. 12
3.10. Safe State Verification. 12
3.11. Memory Characteristics. 12

3.12. Validity Checks. 12
3.13. Operating System (OS) and Run-Time-Executive (RTE). 13
3.14. Operator Interface. 13
3.15. Traceability. 13
3.16. Critical Function Call and Entry Checks. 13
3.17. Command and Data Word Format. 13
3.18. Software Configuration. 13
3.19. Hardware/software Interactions. 13
3.20. Unique Aircraft Requirements. 13
3.21. Unique ICBM Requirements. 14

Chapter 4—DESIGN/EVALUATION CRITERIA CORRELATION MATRIX 15

4.1. Design/Evaluation Criteria Correlation Matrix. 15

Table 4.1. Design/Evaluation Criteria Correlation Matrix. 15

Attachment 1—GLOSSARY OF REFERENCES AND SUPPORTING INFORMATION 17

Attachment 2—GENERIC NUCLEAR SAFETY OBJECTIVES WITH AFMAN 91-118 OR AFMAN 91-119 CROSS-REFERENCE 20

Attachment 3—SUMMARY PRIORITY SCHEME FOR NUCLEAR SAFETY DISCREPANCIES 23

Attachment 4—CAPABILITY MATURITY MODEL (CMM) AND ISO 9001 25

Attachment 5—ADA PRAGMA RESTRICTIONS 30

Chapter 1

GENERAL STANDARDS AND CONTROL

1.1. Terms and Definitions. See **Attachment 1**.

1.2. History and Philosophy. AFR 122-9 and AFR 122-10 previously contained the process and criteria for design certification of nuclear weapon system software and firmware. As regulations were superseded by instructions and manuals, part of AFR 122-9 was absorbed into AFI 91-103 and AFR 122-10 was changed to AFMAN 91-118. However, many of the defining requirements were omitted and some references have changed. MIL-STD-498 superseded DoD-STD-2167 (currently referenced in AFMAN 91-118). Additionally, the software development community has improved processes and tools for mitigating software safety risk. A joint standard (EIA/IEEE J-STD-016) and an international standard (ISO/IEC 12207) are currently used for software development. A tri-service software system safety handbook (Navy lead) and an international standard (IEC 1508) specifically target software safety. The goal of AFMAN 91-119 is to provide guidance for development of safe nuclear weapon system software while allowing application of modern processes for software surety. AFMAN 91-119 consolidates old and new wisdom but is not intended to encourage an isolated view of software surety independent of hardware development (systems view is essential). A separate AFMAN for nuclear software criteria is intended to provide additional visibility and to accommodate rapid advances in software surety. Software requirements in AFMAN 91-118 were moved to AFMAN 91-119. AFSC intends to update this manual approximately once a year with current information.

1.3. Department of Defense (DoD) SafetyStandards. The DoD Nuclear Weapon System Safety Standards form the basis for the safety design and evaluation criteria for nuclear weapon systems. The DoD Nuclear Weapon System Safety Standards state that:

1.3.1. There shall be positive measures to prevent nuclear weapons involved in accidents or incidents, or jettisoned weapons, from producing a nuclear yield.

1.3.2. There shall be positive measures to prevent DELIBERATE prearming, arming, launching, or releasing of nuclear weapons, except upon execution of emergency war orders or when directed by competent authority.

1.3.3. There shall be positive measures to prevent INADVERTENT prearming, arming, launching, or releasing of nuclear weapons in all normal and credible abnormal environments.

1.3.4. There shall be positive measures to ensure adequate security of nuclear weapons, under DoD Directive 5210.41.

1.4. Development Standards. Nuclear systems will comply with a software development standard to ensure a systematic process is used that will increase confidence for successful nuclear certification. Tailoring shall be reviewed and approved by AFSC prior to critical requirements review.

1.4.1. New Systems. New systems will comply with IEEE/EIA 12207.

1.4.2. Existing and Modified Systems. Existing and modified systems will comply with at least one of the following software development standards:

DoD-STD-2167A

MIL-STD-498

EIA/IEEE J-STD-016

IEEE/EIA 12207

1.5. NSA Certification. Codes, coding procedures, software encryption and decryption material with codes, and encryption and decryption programs produced by the NSA receive certification equivalent to the AF process. Therefore, an AF NSCCA or IV&V is not required for design certification of software or firmware certified by NSA.

1.6. DOE Certification. DOE-certified software for use in DOE-provided equipment will not require AF design certification, providing the operating environments are identical. Therefore, an AF NSCCA or IV&V is not required for design certification of software or firmware certified by DOE.

1.7. Air Force Criteria. To comply with the DoD safety standards, the Air Force has implemented a set of minimum design and evaluation criteria for Air Force nuclear weapon systems software. These criteria do not invalidate the safety requirements in other DoD publications, but Air Force activities are required to apply the more stringent criteria. Since the criteria in this manual are not design solutions and are not intended to restrict the designer in the methods and techniques used to meet operational design requirements, they are not all-inclusive. Air Force nuclear weapon system software designers may add feasible and reasonable safety features, as needed. The goal is to design a system that significantly exceeds these safety criteria. These nuclear surety software criteria apply equally to firmware, automata, and other like mechanisms.

1.8. Nuclear Safety Software Categories. Nuclear safety certified software is designated by the NWSSG or AFSC/SEW as software with nuclear safety implications. Software receiving this designation will fall into one of three categories: Cat I, Cat II, and Cat III. Software in categories I and II will be separately safety-certified and listed in T.O. 00-110N-16. Software in category III may, or may not be, separately safety certified; and may, or may not, be subjected to IV&V. AFSC/SEW provides guidance for the scope of the certification effort. Additionally, Discrepancy Reports (DRs) will be generated during NSCCA or IV&V for any errors or deficiencies detected in program design, code, or documentation. Each nuclear safety DR will be recorded and tracked in a discrepancy log that identifies the DR and gives its status. The priority scheme is shown in **Attachment 3**.

1.8.1. Category I. Category I software is software that controls critical function(s) and/or has been designated by the NWSSG or AFSC/SEW as a critical component. NSCCA is required. (Note: Software may be designated as Category I for other reasons; e.g., the software is responsible for the primary security of a nuclear weapon at a remote launch point, the software processes clear text command and control data, etc.)

1.8.2. Category II. Category II software is software that controls critical function(s), but is not designated as a critical component. IV&V is required. Prepare a list of Nuclear Safety Objectives (NSOs) for each nuclear weapon system or modification program. A generic NSO list is shown in attachment 2. All appropriate NSOs must be satisfied in the design and verified by the IV&V.

1.8.3. Category III. Category III software is software that does not control critical function(s), but interfaces with hardware/software which does control critical function(s). Category III software may or may not be subjected to an IV&V, as determined by AFSC/SEW. Category III software includes,

but is not limited to, mission planning system software, and software used to monitor, test, or maintain weapon system interfaces when the warhead or bomb is not mated to or loaded on the delivery system.

1.9. Development Process. The designer of nuclear weapon system software must have a process for both management and engineering activities which is documented, standardized, and integrated into a standard software process for the organization. For software impacting nuclear critical functions, the software development contractor will be certified to at least Level II of the Capability Maturity Model (CMM) or equivalent (such as ISO), and implement a Software Process Improvement (SPI) program. The definitions of various CMM levels is in attachment 4. An organization must coordinate their measurement process with AFSC/SEW for approval. Method/origin of certification (e.g., self evaluation) will be considered during evaluation.

1.10. Requests for Deviations. If the design of Air Force nuclear weapon system software does not meet the requirements contained in this manual, a deviation must be obtained according to the requirements of AFI 91-107. Exceptions to this manual, as evidenced by some current and older designs, do not constitute a precedent to deviate from the criteria.

1.11. Software Advisory Group (SAG). Any system undergoing a software modification to nuclear surety certified software must have a SAG chaired by the engineering major command. This group is typically formed as a subgroup to the nuclear certification working group, nuclear surety working group, or project officers group. The SAG will meet as necessary (determined by the chair, in coordination with HQ AFSC/SEW and the independent engineering evaluation agency).

1.12. Records Disposition. Ensure that all records created by this instruction are maintained and disposed of in accordance with AFMAN 37-139, Records Disposition Schedule.

Chapter 2

DESIGN CRITERIA FOR NUCLEAR WEAPON SYSTEMS SOFTWARE

2.1. Application. These design safety criteria apply to software that receive, store, process, or transmit data to monitor, target, prearm, arm, launch, release, or authorize the use of a nuclear weapon. Design such software to provide the greatest extent of protection against accidental or deliberate unauthorized operation of critical functions.

2.2. Software Specifications. Incorporate applicable nuclear surety design requirements into software specifications (e.g., requirements, design, product specifications). Integrate nuclear surety throughout the software development process including the Request for Proposal (RFP). For new systems a draft NCP must be part of the RFP response from the contractor, and must define which nuclear surety requirements are applicable, and what the nuclear certification approach will be. For a previously certified system, a draft NSIS (Nuclear Surety Impact Statement) must be part of the RFP response from the contractor and must define which nuclear surety requirements are applicable. This NSIS will then be updated as the program is put on contract and the NCP (Nuclear Certification Plan) is developed.

2.3. Higher-Order Language (HOL). Base the software development on an internationally recognized standard (e.g., ISO/IEC-8652:1995, commonly known as Ada 95) that emphasizes application and verification of safety and security. Develop code in assembly or machine language only when the HOL does not provide adequate capability for time-critical or hardware-interfacing functions (justify in the NSIS, NCP, or submit a request for deviation). Use the language of the original source code when modifying critical software.

2.3.1. Pragma Safety Restrictions. The American National Standard for Programming Languages (ANSPL) defines specific prohibitions for software languages that exhibit unacceptable safety behavior. The software developer will comply with applicable safety prohibitions. **C** and Ada are examples of languages with restrictions. Ada pragma restrictions addressed in **Attachment 5** were developed for avionics high integrity software. **C** pragma restrictions are addressed in the industry standard referenced below:

C [Amendment 1: **C** Integrity], ANSI/ISO/IEC 9899-1990/AM 1-1995

2.4. Hierarchical Design. For reducing complexity and to avoid mistakes, design the software in a hierarchical structure. CSUs performing critical functions must be "single purpose" (i.e., one critical function per CSU). The intent of "single purpose" is to avoid untraceable code.

2.5. Fault Detection. Design the software to provide self-check, confidence, or test routines to verify the integrity and proper state of hardware devices that affect or execute critical functions. Design the software to detect critical function failure modes during power-up, operation, and shut-down. Transitory faults (such as corrupted message packets) which do not indicate degraded processing capability shall be detected and dealt with, but do not necessarily need to be reported to the operator. The design shall address the issue of what rate of transitory faults represents a system degradation. This requirement shall not be construed as to permit automatic probing of warhead interfaces, which is prohibited.

2.6. Fault Response. Critical failure modes or attempted illegal entry into a critical function (see paragraph **2.16.**, Critical Function Call and Entry Checks) must result in operator notification of the problem and the status of any automated actions taken. Design the software to revert to a known safe state when a critical function system fault is detected. The software must stop transmitting critical commands upon detecting faults and recycle, self-test, or perform automatic shutdown while displaying associated crew indications.

2.7. Real-Time Software. Software interfacing with critical signals shall be designed such that events associated with nuclear safety critical signals and functions have corresponding response deadlines. These deadlines will determine a fixed interval during which associated processing must be accomplished.

2.8. Responsiveness. Provide techniques for deadlock (waiting for a particular event that will not occur) prevention, detection, or resolution.

2.9. Instruction Alterations. Prevent software from modifying its own instructions or the program instructions of other programs operating within the same system (i.e., no self-modifying code).

2.10. Initialization and Shutdown. Ensure critical function hardware, which is controlled or monitored by software and memory containing nuclear critical information, is initialized or verified to be in a known safe state. Upon system shutdown or program termination, the software must ensure all settable, nonvolatile devices and relays are set to a known safe state.

2.11. Memory Characteristics.

2.11.1. Program Loading and Initialization. Design the system to prevent normal program execution or continuation until all program instructions or data (or both) are loaded and verified. Display the results of the program load verification to system operators. All non-volatile memory must be loaded with executable code, data, or a non-use pattern. If executed as an instruction, this non-use pattern must be detected and dealt with safely. Initialization for memory must be done at system startup or after completion of program and data loading.

2.11.2. Memory Protection. Design memory protection system to ensure operational use does not alter or degrade memory content over time. Storage of critical function programs and data in nonvolatile and read-only memory is preferred. Software must have provisions to ensure erroneous data, resulting from power interrupt, uncorrected memory system error, or other phenomena, do not affect any critical function or component. Software shall not perform error correction on nuclear critical data.

2.11.3. Memory Accessibility. Partition critical function routines and data base elements so that access to these areas is strictly controlled. These areas shall be controlled so that elements or routines which should not have access will be prevented from continuing execution should access to the critical area be attempted.

2.11.4. Memory Declassification. Provide a method to erase or obliterate any clear-text secure codes from memory. Use NSA approved design criteria found in DoDD S-5200.16.

2.12. Validity Checks. Ensure all critical command transmissions originate with manual operator inputs (see paragraph **2.14.**, Operator Interface). Before transmission, the system must verify the state of all

applicable preconditions and inhibits. Do not have the information that defines the prearm unique signal pattern within the software that contains the routine(s) for generating the unique signal. The prearm unique signal shall not be stored in a directly usable form. The prearm unique signal shall only be assembled as a result of a crew member action.

2.13. Operating System (OS) and Run-Time-Executive (RTE).

2.13.1. OS and RTE Development. The OS or RTE must be developed in accordance with a published standard for operating systems (e.g., POSIX, ISO). The OS or RTE should be procured from a commercial source and widely used in a variety of applications. Other factors that will contribute to a presumption of correct and reliable RTE operation are the existence of an active users group for the RTE product and evidence of responsiveness to problem reports by the RTE manufacturer. Ensure only the OS and/or RTE are able to invoke the top level nuclear weapon control computer program component.

2.13.2. OS and RTE Operating Constraints. The OS or RTE shall not continue real-time operation after experiencing either a stack overflow or a hard frame overrun. In either case, manual or automatic restart to a specified safe condition is required. Also, either operator notification of the fault, or logging of the fault is required. Design certification requires that adequate margins be shown to exist such that these conditions cannot occur under the worst case of normal condition processing. A function is normal if it is designed for, even if it is a recovery action for an external device failure.

2.14. Operator Interface. Ensure nuclear critical functions can be terminated by the operator. Ensure nuclear critical functions cannot be initiated with a single action by an operator (two or more actions, such as keystrokes, are required). The software must provide detection and notification of improper operator entries. For authorization and unique signal information software shall take no action on the contents of this information and transmit it unaltered. Operator cancellation of nuclear critical function processing will be accomplished with a single action by an operator.

2.15. Traceability. Each critical nuclear function routine must have a single unique entry point and a structured and complete (all parts of the path must be specified) exit path.

2.16. Critical Function Call and Entry Checks. Ensure only the nuclear weapon control software components are able to call critical function routines and modules. An erroneous entry into a critical function routine or sequence must terminate critical function execution and notify the operator of the error. Ensure critical function command routines cannot be called until all proper conditions exist. All entries into nuclear critical function routines must have checks to ensure such entries are both authorized and approved (operator action is accomplished and all preconditions are met).

2.17. Command and Data Word Format. Select decision logic data values that require a specific binary data pattern of "ones" and "zeros" (not all "ones" or all "zeros") to reduce the likelihood of hardware or software malfunctions that satisfy the decision logic for critical function initiation or propagation. Data patterns will be selected such that a single bit flip will not result in a valid command or data word.

2.18. Software Configuration. Evaluation of software for compliance with nuclear surety requirements shall be performed on a specific configuration of final executable code.

2.19. Hardware/software Interactions. The design agent will present all known failure modes of interfacing hardware. The agent will design or implement measures to ensure that each failure mode is recognized and dealt with in a safe manner.

2.20. Unique Aircraft Requirements.

2.20.1. Command Verification Protocol. For aircraft and air-launched nuclear weapons, ensure the verification of critical commands transferred to remote units for processing or execution. If a non-transitory communication error occurs, the command sequence must be reset to its initial state.

2.20.2. In-flight Reversible Lock. If the inflight reversible lock is under software control, design the software controlling release or launch of a nuclear weapon with a unique control or control setting for locking and unlocking the in-flight reversible lock. Make this control separate from the release and launch controls and the release consent.

2.20.3. Prearm Consent. The design may implement prearm consent through software inhibits and controls. However, the consent signal must originate only through crew action. Removal of prearm consent must result in terminating the prearm functions in process and must inhibit prearm until consent is reestablished. Any change in consent status must also be sent to the weapon, which will then inhibit any critical function processing under weapon system control.

2.21. Unique ICBM Requirements. Crewmember initiation of critical functions requires separate and independent action.

Chapter 3

EVALUATION CRITERIA FOR NUCLEAR WEAPON SYSTEMS SOFTWARE

3.1. Evaluation Program. The nuclear surety impact statement (NSIS) and/or nuclear certification plan (NCP) required by AFI 91-103 must define tests and analyses to verify compliance with the design criteria of **Chapter 2**. The NSIS will define applicability of design criteria to each software component.

3.2. Software Standards Compliance. Evaluate software design documentation for MIL-STD 498, MIL STD 2167, J-STD-016, or IEC 12207 compliance and incorporation of applicable nuclear surety design requirements. Compliance with old standards is permitted for minor (as determined by AFSC) software modifications.

3.3. Critical Data Element Analysis. Define the CSUs that change or evaluate critical function data elements (such as constants, variables, and flags). Define the chronological sequence of data element changes and evaluations. Define the minimum conditions required for critical function execution (sensitivity analysis). Use this information to evaluate the HOL safety features and estimate a probability for hardware induced software problems (inadvertent activation of critical functions) according to AFI 91-107, Table 2.

3.3.1. Pragma Safety Restrictions. Verify applicable pragmas prohibited by ANSPL or **Attachment 5** are not used.

3.4. Hierarchical Design. Determine presence of hierarchical structure. Evaluate critical function CSUs for single purpose.

3.5. Fault Detection. Demonstrate the self-check, confidence, or test routines. Demonstrate the fault detection capability of the system during power-up, operation, and shut-down. This may be accomplished with an analysis if demonstration could activate a critical function. AFSC/SEW must approve failure modes selected for fault insertion. The design documentation shall enumerate the fault conditions detected. All reasonable error conditions shall be included. Fault conditions shall be designated as transitory or system degrading and handled appropriately.

3.6. Fault Response. Demonstrate the software response to detected system faults. The demonstration must show upon fault detection:

reversion to known safe state

transmission of critical commands halted

recycle, self-test, or perform automatic shutdown

operator notification

status of automated actions

3.7. Real-Time Software. Documentation for real-time software shall include specification of fixed deadlines for service of events associated with nuclear critical signals and functions. Documentation shall also include sufficient information about scheduling protocols, task priorities and such other information as may be necessary to rigorously prove that such deadlines will be met under all system load and inter-

rupt conditions (e.g., rate monotonic analysis). Certification shall be contingent on determination that deadlines are consistent with system nuclear surety requirements.

3.8. Responsiveness. Demonstrate techniques for deadlock prevention, detection, or resolution. Introduce a deadlock condition and monitor for appropriate response.

3.9. Alterations Detection. Software containing any self-modification of design, code, or critical data is prohibited. The method of detection depends on the type of verification:

3.9.1. NSCCA - Develop or select a software tool to detect self-modification or modification of ancillary programs.

3.9.2. IV&V - Check the design and code (if available) for any indication of self-modification.

3.10. Safe State Verification. Identify critical function hardware and corresponding known safe states. Verify critical function hardware is in a known safe state at software initialization and shutdown.

3.11. Memory Characteristics.

3.11.1. Program Loading and Initialization. Demonstrate that errors will be detected and operators notified; reset or synchronization functions are operable; automatic operation will not start until all valid and correct data are loaded and verified; reloading or changing that part of memory involving critical functions will stop unless the proper means for entry is used; improper reload or change will be rejected and indicated to the operator; and the block of proper program data or instructions to be transferred fills each section of memory.

3.11.2. Memory Protection. Self-modification cannot be measured but several design practices can protect against it. Assurance can be achieved in many different ways; these are only a few examples:

3.11.2.1. Use programming languages that prevent self-modification of code

3.11.2.2. Use an encapsulation kernel (a method used in security to prevent access of one program to another)

3.11.2.3. Use operating systems that include such protection features

3.11.2.4. Put code in read-only memory

3.11.2.5. Separate the critical code from other code (build in firewalls or put it on a different computer)

3.11.2.6. Rational Apex and Rational Rose tools for Ada 95 have some capability to avoid modification of its own instruction set or instructions of other programs.

3.11.3. Memory Accessibility. Provide an analysis showing what percentage of the software can access the critical areas and a description of how the memory access control system works.

3.11.4. Memory Declassification. Demonstrate the method used to erase or obliterate clear-text secure codes from memory. The National Security Agency must approve all declassification procedures.

3.12. Validity Checks. Identify all critical command transmissions and their preconditions and inhibits. Determine method of origination. Determine process for preventing transmission if preconditions and

inhibits are not satisfied. The weapon system or modification will not be certified if software generates critical command transmissions without manual operator input and satisfaction of preconditions and inhibits. Verify the prearm unique signal is not stored in a directly usable form. Verify the usable prearm unique signal can only be assembled as a result of a crew member action. Identify software that contains the routine(s) for generating the unique signal. Verify the routine(s) is/are initiated through crewmember action. Show that the unique signal generator software does not contain the prearm unique signal pattern.

3.13. Operating System (OS) and Run-Time-Executive (RTE). Develop a statistically valid test to demonstrate stable operation in a variety of worst-case operating conditions. Any failures (evidenced by overflow, overwrite, or non-return) must be corrected and the system successfully retested prior to certification.

3.14. Operator Interface. Demonstrate operator input error detection and handling affecting critical functions. The demonstration must show the software will notify the operator and will not enable any critical function.

3.15. Traceability. Each critical function must be addressed in the safety analysis, showing the single entry point and all planned exit paths.

3.16. Critical Function Call and Entry Checks. Confirm the presence of a security function in each critical function routine that identifies unauthorized entries. Demonstrate termination and notification following unauthorized entry identification.

3.17. Command and Data Word Format. NSCCA, IV&V, or V&V will check for disallowed bit patterns.

3.18. Software Configuration. Evaluation of software for compliance with nuclear surety requirements should be done on as high a level as possible subject to the condition that correspondence with executed code is unambiguous. Compiled high level languages such as Ada generally meet this requirement. In situations where code is automatically generated from graphical models or other higher level construct (such as those associated with Object Oriented Analysis), such models may be evaluated if they can be presented in a human readable format and if the translation rules are sufficiently well explained that correspondence with executable code is unambiguous.

When there exists design information from which programs are developed by manual processes, inclusion of this information for reference purposes is encouraged, but cannot form the sole basis of evaluation. In no case shall software be certified based on evaluation of out-of-date code; an update to the executable requires an update to the code evaluated.

3.19. Hardware/software Interactions. Verify that there are no unidentified failure modes and that there are handlers for each failure mode.

3.20. Unique Aircraft Requirements.

3.20.1. Command Verification Protocol. Identify critical commands transferred to remote units for processing or execution. Define verification scheme to be used. Demonstrate communication error

detection method (see paragraph **3.5.**, Fault Detection) and reset process following error detection (see paragraph **3.6.**, Fault Response).

3.20.2. In-flight Reversible Lock. Identify any software controlling release or launch of a nuclear weapon. Define the programming scheme that provides a unique control or control setting for locking and unlocking the in-flight reversible lock. Verify the control is separate from the release and launch controls and the release consent.

3.20.3. Prearm Consent. If prearm consent is used, verify the software accomplishes the following:

crew action origination

termination following consent removal

prearm inhibited until reestablished

critical function inhibited following change in consent status

3.21. Unique ICBM Requirements. Demonstrate software constraint that prohibits critical function initiation without dual input.

Chapter 4

DESIGN/EVALUATION CRITERIA CORRELATION MATRIX

4.1. Design/Evaluation Criteria Correlation Matrix. This table shows the correlation between the previous requirements in AFMAN 91-118 and this current verson; it also shows the correlation between the design criteria and the evaluation criteria.

Table 4.1. Design/Evaluation Criteria Correlation Matrix.

91-119 Design Criteria	91-119 Evaluation Criteria	Original 91-118 Requirements
2.1 Application	3.1 Evaluation Program	2.12, 4.2.2
2.2 Software Specifications	3.2 Software Standards Compliance	2.12.1, 4.2.2.1
2.3 Higher-Order Language (HOL)	3.3 Critical Data Element Analysis	2.12.2, 4.2.2.2
2.3.1 Pragma Safety Restrictions	3.3.1 Pragma Safety Restrictions	new
2.4 Hierarchical Design	3.4 Hierarchical Design	2.12.3
2.5 Fault Detection	3.5 Fault Detection	2.12.4, 4.2.2.5/6
2.6 Fault Response	3.6 Fault Response	2.12.5, 4.2.2.4
2.7 Real-Time Software	3.7 Real-Time Software	2.12.6
2.8 Responsiveness	3.8 Responsiveness	2.12.7
2.9 Instruction Alterations	3.9 Alterations Detection	2.12.8
2.10 Initialization and Shutdown	3.10 Safe State Verification	2.12.9
2.11.1 Program Loading and Initialization	3.11.1 Program Loading and Initialization	2.14, 2.13.6, 4.2.2.3.2
2.11.2 Memory Protection	3.11.2 Memory Protection	2.13.1/2/3, 4.2.2.3.1
2.11.3 Memory Accessibility	3.11.3 Memory Accessibility	2.13.4
2.11.4 Memory Declassification	3.11.4 Memory Declassification	2.13.5, 4.2.2.3.3
2.12 Validity Checks	3.12 Validity Checks	2.14.2, 2.34.1.1
2.13.1 OS & RTE Development	3.13 OS & RTE	2.15
2.13.2 OS & RTE Operating Constraints	3.13 OS & RTE	2.15
2.14 Operator Interface	3.14 Operator Interface	2.16.1, 4.2.2.7
2.15 Traceability	3.15 Traceability	2.16.2
2.16 Critical Func Call and Entry Checks	3.16 Critical Func Call and Entry Checks	2.16.3
2.17 Command and Data Word Format	3.17 Command and Data Word Format	2.16.4
2.18 Software Configuration	3.18 Software Configuration	new

2.19 HW/SW Interactions	3.19 HW/SW Interactions	new
2.20.1 Command Verification Protocol	3.20.1 Command Verification Protocol	2.14.1
2.20.2 In-Flight Reversible Lock	3.20.2 In-Flight Reversible Lock	2.34.2.2
2.20.3 Prearm Consent	3.20.3 Prearm Consent	2.34.1.2.2
2.21 Unique ICBM Requirements	3.21 Unique ICBM Requirements	2.16.1

FRANCIS C. GIDEON, JR., Maj Gen, USAF
Chief of Safety

Attachment 1

GLOSSARY OF REFERENCES AND SUPPORTING INFORMATION

References

INDUSTRY REFERENCES

IEC 1508/1513, *Safety Related SW / Nuclear*

ISO/IEC 8652, Ada 95 (Annex H: Safety & Security)

IEEE Std 1228, *SW Safety Plans*

IEEE Std 982.1/2, *Standard Measures to Produce Reliable Software*

EIA/IEEE J-STD-016, *SW Development*

US 12207, *SW Development*

UL 1998, *Standard for Safety Related SW*

Safeware: System Safety and Computers, Leveson, 1995, Addison-Wesley

5th Int Symposium on SW Reliability Engineering, Methodology of Independent Software Nuclear Safety Analysis, Nov 94

GOVERNMENT REFERENCES

DoD 3150.2-M, *DoD Nuclear Wpn System Safety Standards, Policy, and Criteria*

DoD Guidelines for SW Dev

MIL-STD-498, *SW Development*

MIL-STD-882, A.8.7, *SW Mishap Risk Assessment Process*

MIL-HDBK-272, 5.6, *Nuclear Weapon System*

Data Processing HW & SW

MIL-STD-1822, 4.5.12, *Automata and SW*

NASA-STD-2201-93, *Software Surety*

NRC HICB-14, *Guidance on SW Reviews*

USAF REFERENCES

AFI 91-101, *AF Nuclear Weapons Surety Program*

AFI 91-103, *AF Nuclear Safety Certification Program*

AFI 91-107, *Design, Evaluation, Troubleshooting, and Maintenance Criteria for Nuclear Weapon Systems (implements AFMANs 91-118/119)*

AFMAN 91-118, *Safety Design and Evaluation Criteria for Nuclear Weapon Systems Hardware*

AFR 122-9, *Nuclear Surety Design Certification Program for Nuclear Weapon System Software and Firmware (superceded)*

AFWL SW Evaluation Manual

AFISC System Safety Handbook 1-1, SW System Safety

AFOTEC PAM 99-102, Vol 6, *SW Maturity Eval Guide*

SW Criteria for Nuclear Safety Certification, SA-ALC/SWPE, 54899-F141-UT-01

Abbreviations and Acronyms

AF—Air Force

AFI—Air Force Instruction

AFMAN—Air Force Manual

AFR—Air Force Regulation

AFSC—Air Force Safety Center

ANSPL—American National Standard for Programming Languages

CMM—Capability Maturity Model

CSCI—Computer Software Configuration Item

CSC—Computer Software Component (subset of a CSCI)

CSU—Computer Software Unit (separately testable element of a CSC)

DoD—Department of Defense

DOE—Department of Energy

DR—Discrepancy Report

EIA—Electronic Industries Alliance

HOL—Higher Order Language

HW—Hardware

ICBM—Intercontinental Ballistic Missile

IEC—International Electrotechnical Commission

IEEE—Institute of Electrical and Electronics Engineers

ISO—International Organization for Standardization

IV&V—Independent Validation and Verification

KPA—Key Process Area

NCP—Nuclear Certification Plan

NSA—National Security Agency

NSCCA—Nuclear Safety Cross Check Analysis

NSIS—Nuclear Surety Impact Statement

NSO—Nuclear Safety Objective

NWSSG—Nuclear Weapons System Safety Group

OS—Operating System

POSIX—Portable Operating System Interface

RFP—Request For Proposal

RTE—Run-Time Executive

RTS—Run-Time System

SEW—Weapons, Space, and Nuclear Safety Division

SPI—Software Process Improvement

STD—Standard

SW—Software

V&V—Validation and Verification

Attachment 2

GENERIC NUCLEAR SAFETY OBJECTIVES WITH AFMAN 91-118 OR AFMAN 91-119 CROSS-REFERENCE

The **authorization** function will (AFMAN 91-118, 2.2.2.1):

1. Authorize use of the weapon system

2. Prevent prearming, arming, or launching without prior authorization (examples of designs include Permissive Action Link and the Minuteman enable device).

3. Be implemented using the information control concept.

4. Not prevent safing regardless of the state of authorization device.

5. Be reversible.

The **prearming** function will (AFMAN 91-118, 2.2.2.2):

6. Permit arming.

7. Preclude prearming in the absence of a prearm command signal.

8. Prevent arming if the prearming device is bypassed.

9. Isolate the prearming signal from the authorization function.

10. Be derived from some part of the weapon system that is under direct human control.

11. Be reversible up to the time of launch.

12. Use a uniquely coded signal.

13. Be physically unavailable until its use is required.

The **launching** function will (AFMAN 91-118, 2.2.2.3):

14. Permit operation of the propulsion system.

15. Be controlled with two independent functions (e.g., booster arm/safe and ignition commands).

16. Prevent ignition when device has been "safed."

17. Preclude unauthorized/accidental transmission of booster arm and ignition commands.

18. Have a reversible propulsion system ignition coding device.

The **arming** function will (AFMAN 91-118, 2.2.2.5):

19. Permit warhead detonation through a selected fuse signal (e.g., radar, contact, and timer).

20. Prevent detonation if the arming function has been bypassed.

21. Preclude arming prior to measurement of the proper operational environment; this measurement must include a "good guidance" signal.

22. Prevent erroneous transmission of the proper operational environment signal.

The **targeting** function will (AFMAN 91-118, 2.2.2.6):

23. Prevent accidental/deliberate changes to targeting.

24. Display to the launch control point operators any changes to targeting data/functions which occur prior to launch.

25. Prevent nuclear detonation except within boundaries of the designated target area (by preventing a "good guidance" signal).

Single component failures will (AFMAN 91-118, 2.10):

26. Not cause the system to be in an authorized, prearmed, or armed state.

27. Not cause the inadvertent transmission/operation of the critical functions.

Human engineering will (AFMAN 91-118, 2.11):

28. Be used to add features that either eliminate human error and unauthorized acts or limit their consequences.

29. Prevent any two independent human errors from causing the authorization, prearming, arming, or launching of the weapon system.

30. Incorporate positive measures to prevent deliberate, unauthorized, or accidental operations of the weapon system that could degrade nuclear safety.

31. Eliminate/minimize dependency of the safety and security of the weapon system on administrative procedures.

The **launch control system** will (AFMAN 91-118, 2.26):

32. Permit launching only through the intentional operation of the authorization and launch control function/devices.

33. Be the only system which will be able to authorize/start a launch sequence, launch a missile, or operate the propulsion system.

34. Remain in, or return to, a safe state when component failure occurs.

35. Not implement the prearming and safing functions as complementary functions (i.e., the absence of prearming will not be construed as safe and vice versa).

The **monitor system** will (AFMAN 91-118, 2.28):

36. Provide the operator continuous or on-demand monitoring of the safety status of the missile’s propulsion system, warhead (if the warhead has monitoring circuits), reentry vehicle/system, and launch control system.

37. Provide the operator with positive indications of any changes to the safety status of the monitored systems.

38. Automatically remove power if an unsafe condition is detected.

The **command, control, and communications system** will (AFMAN 91-118, 2.29):

39. Use secure codes to authorize the launch of or to launch a missile.

40. Not allow the authorization or launching of a missile by a single person; at least two people must actively cooperate to command authorization and launch, even after secure codes are available .

41. Prevent bypass or unauthorized readout of the secure code devices.

42. Control access to the code storage devices to prevent unauthorized code changes.

43. Implement the prearming and launching commands as unique signals.

44. Not store the prearming and launching signals at the launch point in a directly usable form.

45. Prevent unauthorized use of the prearming and launching signals (implementation examples: derive signals from secure codes; store signals in permuted form; store parts of the signals in separate locations).

46. Allow one or more missiles to be launched without revealing or compromising the codes for other missiles.

47. Allow one or more launch control points to monitor and take compensatory action if an unauthorized critical command message or status is detected.

Automata and software will (AFMAN 91-119):

48. Not be made up of memory whose contents alter or degrade over a period of time.

49. Have provisions to prevent unexpected changes in volatile memory from adversely impacting a critical function.

50. Prevent a single hardware failure from causing a memory change that could initiate a critical function.

51. Verify and validate the correct loading of all data and programs.

52. Prevent unauthorized changes to memory.

53. Notify the launch control point operator or maintenance/coding personnel of errors detected during authorized memory loads/changes.

54. Detect, inhibit, and report (to the operator) any unauthorized attempts at loading/changing critical functions in memory.

55. Ensure that each section of memory is filled by the block of proper program data or instructions to be transferred. This exact-fill requirement can be met by tailoring data for exact fill or using filler bits. If this requirement cannot be met, the memory will be overwritten or cleared before writing the transferred data or program instructions.

56. Declassify all clear-text secure codes through erasure/obliteration after their authorized use.

57. Verify/validate all critical commands prior to transmission.

58. Inhibit transmission of any critical command found to be in error and notify the operator of the error.

59. Not be able to bypass operator control of critical functions.

60. Detect erroneous entries into critical routines/functions and immediately recycle to the proper sequence, self-test mode, or automatic shutdown.

61. Adhere to top-down design decomposition and structured programming methodology.

62. Provide self-check, confidence, or test routines of all critical hardware /circuits.

63. Require a method permitting only authorized entry to the memory when reloading or changing memory involving critical functions.

Attachment 3

SUMMARY PRIORITY SCHEME FOR NUCLEAR SAFETY DISCREPANCIES

A3.1. Critical (Priority 1). An error which could lead to a violation of one or more of the DOD Nuclear Weapon System Safety Standards or compromise one or more of the critical functions. Priority 1 discrepancies should be directly traceable to certain and/or positive violations of at least one of the DOD safety standards or to inadvertent or unauthorized execution of any one of the critical functions (as specified in AFMAN 91-118). These discrepancies must be corrected prior to granting certification. Workarounds are not acceptable.

A3.2. Urgent (Priority 2). An error which could lead to a violation of one of the DOD safety standards, with a small probability of occurrence (within a credible but abnormal environment, including single-point component failures). Priority 2 discrepancies have a serious nuclear safety impact, but they require a low probability event or combination of events in order for the error(s) to occur. Note that a credible probability must be no smaller than that for a single component failure. Such a problem may not be corrected by an administrative workaround. Resolution of all errors in this category must be corrected prior to granting certification.

A3.3. Degraded (Priority 3). An error which meets one of two conditions:

A3.3.1. The degradation of, but not elimination of, a nuclear safety-related safeguard (must define degree of acceptable degradation).

A3.3.2. An otherwise Priority 2 discrepancy with a workaround already in place. Priority 3 discrepancies can lead to a safety degradation but will not result in critical function compromise, or the discrepancies could instead reflect a critical or serious problem which has been temporarily corrected via workaround. (An example would be an event that could be recorded as passed but yet could not lead to warhead detonation because of a subsequent check.) Priority 3 errors are "high" candidates for correction but may be tolerated (in the system) while an update is pending (i.e., this error may be deferred but not ignored).

A3.4. Noncritical (Priority 4). A technical noncompliance violation of AFMAN 91-119 which cannot be directly related/traced to a compromise of the DOD safety standards or critical functions. Priority 4 discrepancies are technical violations of AFMAN 91-119, with minimal or unquantifiable impact. (For example, a coding error that results in an erroneous or undefined display but cannot lead to operator error.) These problems are recommended for correction, but they may exist in the system indefinitely. Note, however, that too many errors in this category may require that some corrections be made, particularly when the cumulative effects are likely to impair operator capability (i.e., "many" Priority 4 errors may equal the effects of "one" Priority 2 error).

A3.5. Minor (Priority 5). An error which meets one of two conditions: a. Documentation errors which could be a safety problem if implemented in the code, but the code is otherwise correct. b. Code errors which have no current impact. Priority 5 discrepancies are typically either minor documentation errors or code errors with no present definable impacts. If the code and documentation are both wrong, or if the error is found before the code is produced, then the priority of the error(s) is based on the assumption that the code will (or could) ultimately reflect that error. An extraneous code which could have a safety impact

if executed, but which cannot be executed, is also a Priority 6 error. These errors are also recommended for correction, but they may remain in the system indefinitely. An example of a code which could have a (higher priority) safety impact would be a wrong value but, because of a wrong address, that **value** is never used. (If the address was nonexistent, the error would remain Priority 5.)

Attachment 4

CAPABILITY MATURITY MODEL (CMM) AND ISO 9001

A4.1. The Capability Maturity Model for Software.

The Capability Maturity Model for Software describes the principles and practices underlying software process maturity and is intended to help software organizations improve the maturity of their software processes in terms of an evolutionary path from ad hoc, chaotic processes to mature, disciplined software processes. The CMM is organized into five maturity levels. A maturity level is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement.

A4.2. The Five Maturity Levels.

The following characterizations of the five maturity levels highlight the primary process changes made at each level:

1) Initial The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.

2) Repeatable Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

3) Defined The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

4) Managed Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

5) Optimizing Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

A4.3. Key Process Areas (KPAs).

Except for level 1, each maturity level is decomposed into several key process areas that indicate the areas an organization should focus on to improve its software process. Key process areas identify the issues that must be addressed to achieve a maturity level. Each key process area identifies a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability. The key process areas and their purposes are listed below. The name of each key process area is followed by its two-letter abbreviation. By definition there are no key process areas for level 1.

LEVEL 2. The key process areas at level 2 focus on the software project's concerns related to establishing basic project management controls, as summarized below:

Requirements Management (RM)

Establish a common understanding between the customer and the software project of the customer's requirements that will be addressed by the software project.

Software Project Planning (PP)

Establish reasonable plans for performing the software engineering and for managing the software project.

Software Project Tracking and Oversight (PT)

Establish adequate visibility into actual progress so that management can take effective actions when the software

project's performance deviates significantly from the software plans.

Software Subcontract Management (SM)

Select qualified software subcontractors and manage them effectively.

Software Quality Assurance (QA)

Provide management with appropriate visibility into the process being used by the software project and of the products being built.

Software Configuration Management (CM)

Establish and maintain the integrity of the products of the software project throughout the project's software life cycle.

LEVEL 3. The key process areas at level 3 address both project and organizational issues, as the

organization establishes an infrastructure that institutionalizes effective software

engineering and management processes across all projects, as summarized below:

Organization Process Focus (PF)

Establish the organizational responsibility for software process activities that improve the organization's overall

software process capability.

Organization Process Definition (PD)

Develop and maintain a usable set of software process assets that improve process performance across the projects and provide a basis for cumulative, long-term benefits to the organization.

Training Program (TP)

Develop the skills and knowledge of individuals so they can perform their roles effectively and efficiently.

Integrated Software Management (IM)

Integrate the software engineering and management activities into a coherent, defined software process that is

tailored from the organization's standard software process and related process assets.

Software Product Engineering (PE)

Consistently perform a well-defined engineering process that integrates all the software engineering activities to

produce correct, consistent software products effectively and efficiently.

Intergroup Coordination (IC)

Establish a means for the software engineering group to participate actively with the other engineering groups so the

project is better able to satisfy the customer's needs effectively and efficiently.

Peer Reviews (PR) Remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software work products and of the defects that can be prevented.

LEVEL 4. The key process areas at level 4 focus on establishing a quantitative understanding of both the software process and the software work products being built, as summarized below:

Quantitative Process Management (QP)

Control the process performance of the software project quantitatively.

Software Quality Management (QM)

Develop a quantitative understanding of the quality of the project's software products and achieve specific quality

goals.

LEVEL 5. The key process areas at level 5 cover the issues that both the organization and the projects must address to implement continuous and measurable software process improvement, as summarized below:

Defect Prevention (DP)

Identify the causes of defects and prevent them from recurring.

Technology Change Management (TM)

Identify beneficial new technologies (i.e., tools, methods, and processes) and transfer them into the organization in

an orderly manner.

Process Change Management (PC)

Continually improve the software processes used in the organization with the intent of improving software quality,

increasing productivity, and decreasing the cycle time for product development.

A4.4. Mapping ISO 9001 to the CMM.

The Capability Maturity Model for Software (CMM), developed by the Software Engineering Institute, and the ISO 9000 series of standards, developed by the International Standards Organization, share a common concern with quality and process management. The two are driven by similar concerns and intuitively correlated. Although an ISO 9001-compliant organization would not necessarily satisfy all of the level 2 key process areas, it would satisfy most of the level 2 goals and many of the level 3 goals. Because there are practices in the CMM that are not addressed in ISO 9000, it is possible for a level 1 organization to receive ISO 9001 registration; similarly, there are areas addressed by ISO 9001 that are not addressed in the CMM. A level 3 organization would have little difficulty in obtaining ISO 9001 certification, and a level 2 organization would have significant advantages in obtaining certification. There are 20 clauses in ISO 9001, which are summarized and compared to the practices in the CMM. There is judgment involved in making this comparison, and there are differences in interpretation for both ISO 9001 and the CMM.

Table A4.1. CMM & ISO 9001 Correlation.

CMM Key Process Area	CMM Activities Covered Under 9001 Basic Interpretation	CMM Activities Covered by 9001 Judgement of Auditor	CMM Activities Not Covered by 9001
CMM Level 2 KPAs			
Rqmts Mgmt	1,3		2
SW Proj Planning	3,5,6,7,11,12,14	1,4,8,13	2,9,10,15
SW Proj Tracking & Oversight	2,11,12,13	1,3,4,5,6,7,8,9,10	
SW Subcontract Mgmt	1,2,3,12	6,8,10	4,5,7,9,11,13
SW QA	1,2,4,5,6,7	8	3
SW Config Mgmt	1,2,4,5,6,7,8,9	10	3
CMM Level 3 KPAs			
Org Process Focus		1,2,3	4,5,6,7
Org Process Def		1,5,6	2,3,4
Training Pgm	1,2,6	5	3,4
Integrated SW Mgmt		3,10,11	1,2,4,5,6,7,8,9
SW Product Eng	1,5,6,7,8,9,10	2,3,8	4
Intergroup Coord	2,3	1,4,5,6,7	
Peer Reviews	2,3		1
CMM Level 4 KPAs			
Quantitative Process Mgmt		1,3,5	2,4,6,7
SW Quality Mgmt	3	1,2	4,5
CMM Level 5 KPAs			
Defect Prevention		1,3,4,5,6,7	2,8
Tech Change Mgmt		1,7,8	2,3,4,5,6
Process Change Mgmt		3,9	1,2,4,5,6,7,8,10

Attachment 5

ADA PRAGMA RESTRICTIONS

A5.1. References.

a. ANSI/MIL-STD-1815A-1983, *Ada Programming Language*

b. *Ada Quality and Style: Guidelines for Professional Programmers,* Software Productivity Consortium, 1989

c. ARINC-613, *Guidance for Using the Ada Programming Language in Avionics Systems,* Aeronautical Radio, Inc., Jan 88

d. CMU/SEI-87-TR-9, *Ada Adoption Handbook: A Program Manager's Guide,* SEI, 1987

e. CMU/SEI-89-TR-13, *Ada Adoption Handbook: Compiler Evaluation and Selection,* SEI, 1989

A5.2. Ada Development Standards.

The requirements of this section are intended to promote the development of software that is documented, is reliable, and will meet *performance* and certification requirements.

A5.2.1. Types, Pragmas, and Clauses.

The requirements of this section are intended to specify the features of Ada *types pragmas,* and *clauses to* be used in the software to meet the objectives of this standard.

a. For each *data declaration* for which the *bit mapping* may affect results, *representation clauses* shall be used.

b. For each representation clause, the bit mapping requirements shall be documented in *source code comments* physically local to the clause declaration.

c. The software application shall not change the *value* of the *constants* of *package SYSTEM.*

d. In the source code, only those *compilation units* required for execution of a compilation unit shall be made visible (using a *WITH clause)* to each unit.

e. *RENAME declarations,* rather than the *USE clause,* shall be used in the source code to overload infix *operators* or shorten entity names.

f. Only Ada *attributes* defined in Reference B shall be used in the source code.

g. If a deviation is granted for the use of *assembly language* code in the software application, pragma *INTERFACE* shall be used in the source code to include it.

h. For each specific use of pragma *SUPPRESS* in the source code, physically local source code comments shall document justification for its use and its impact on the software application.

i. For each specific use of the pragma *lNLlNE,* physically local source code comments shall document the impact of its use on *memory* usage and *execution* speed.

j . If function UNCHECKED_CONVERSION is used, physically local source code comments shall document its impact on the computational results for each use.

k. For each specific use of pragma *PACK,* physically local source code comments shall document the impact of the pragma's use on the software application.

1. For each specific use of attribute *ADDRESS* or type *SYSTEM.ADDRESS,* physically local source code comments shall document justification for the use and its impact on the software application.

m. For each specific use of RENAME declaration, physically local source code comments shall document the *scope* of the use.

A5.2.2. Tasking, Memory Management, and Exceptions.

These requirements are intended to promote software that meets reliability requirements, is *deterministic,* and complies with certification requirements. Deterministic defines the capability of a software program, given the identical inputs at the identical times and the same preconditions for a specified time frame, to have the same control flow and to produce identical outputs at the identical times for the time frame.

a. Ada *tasks* shall be elaborated only at *system initialization.*

b. The software application shall not *deallocate* memory.

c. If *generics* are used in the software application, they shall be *instantiated* only during system initialization.

d. Ada *exception* processing (that is, raising and handling) in the source code shall be traceable to software requirements.

e. Task entry points shall not be used for interrupt processing.

A5.2.3. Organization and Notation.

The requirements of this section are intended to promote source code that is consistently organized and *accesses* other compilation units in a consistent manner.

a. *Implementation* dependent source code of the software application shall be placed in compilation units separate from other Ada source code compilation units.

b. When a compilation unit is made visible with the use of a WITH clause in the source code of a compilation unit, *qualified (dot) notation* shall be used in the source code for referencing any entity of other units.

c. The number of bits for a system storage unit shall be represented in the source code using the constant of package SYSTEM, *STORAGE_UNIT.*

d. ASCII *control characters* shall be referenced in the source code according to their definitions in the Ada package *STANDARD.ASCII.*

e. The software application shall not use the package *MACHINE_CODE.*

f. The use of generic *formals* and restrictions on generic *parameters* shall be documented in physically local source code comments.

A5.2.4. Precision, Accuracy, and Numerics.

These requirements are intended to promote correct and accurate computations by the software.

a. *DIGITS* specifications shall be used in the source code to provide the required degree of *accuracy,* rather than the default accuracy of predefined types, such as FLOAT or INTEGER.

b. *Named numbers* shall be used in source code declarations when defining numeric constants.

A5.2.5. Delivery.

The requirements of this section are an elaboration of the requirements of Reference A for the deliverables associated with Ada software development and certification.

a. The *Version Description Document* (VDD) for Ada software shall include or reference the RTS specification, which documents the constraints, limitations, and implementation support for Ada clauses, pragmas, and other implementation dependent features of the RTS.

b. The VDD for Ada software shall include or reference the Ada compiler specification, which includes identification of the compiler's name, version, validation number, validation date, constraints, limitations, and support for implementation dependent characteristics as listed in Appendix F of Reference B.

c. The VDD shall reference and the software *delivery* shall include a list of the implementation dependent characteristics of package *STANDARD* and the specification of package SYSTEM.

d. The VDD shall reference and the software delivery shall include the specifications of compilation/ library *units* which are not predefined by Reference B but are used by the software application (for example, math packages or interface packages), including a listing of implementation dependent characteristics of each unit.

A5.2.6. Supplier Ada Standard Content Requirements.

The requirements for the suppliers avionics standard of this section are an elaboration of the requirements of Reference A for the *Software Development Plan.* The supplier Ada standard shall include the identification of:

a. Ada timing and scheduling features to be used, the standards and conventions to be used in the design of the timing/scheduling model and implementing the model, the notation and rules to be used in the timing/scheduling model, and the analyses to be performed on the timing/scheduling model and implementation to ensure the software end item will meet the performance requirements.

b. Ada *tasking* features to be used, the standards and conventions to be used in the design and programming of Ada tasking, the notation and rules to be used in modeling Ada tasking, and the analyses to be performed on the tasking model and tasking implementation to ensure the reliability and integrity of the implementation.

c. Ada memory management (memory *allocation* and *deallocation,* including the use of Ada *access types)* features to be used, the standards and conventions to be used in the design and programming of Ada memory management features, the notation and rules to be used in modeling Ada memory management, and the analyses to be performed on the memory management model and implementation to ensure the reliability and integrity of the implementation.

d. Ada exception processing features to be used, the standards and conventions to be used in the design and programming of Ada exception processing, the notation and rules to be used in modeling Ada exception processing, and the analyses to be performed on the exception processing model and implementation to ensure the reliability and integrity of the implementation.

e. Ada types to be used and identification of the analyses (for example, dimensional analysis) to be performed on the use of the types to ensure computational reliability, functional acceptability, and memory usage.

f. Ada pragmas to be used and identification of the analyses (in terms of impact on reliability, timing, memory usage, memory access, and verifiability) to be performed on the use of each Ada language pragma.

g. Ada clauses to be used and identification of the analyses (in terms of impact on reliability, timing, memory usage, memory access, and verifiability) to be performed on the use of each Ada clause.

h. standards, conventions, and processes to be used in analyzing the Ada RTS (or the parts of the RTS *loaded* into the end item) to ensure it is deterministic, is functionally correct, and will satisfy certification requirements.

i. standards and conventions to be used for declarations of Ada types, including constants and *variables* (including conventions for specifying the accuracy, *precision,* and range of the types), for declarations of packages and *subprograms,* and for declarations of data structures, including *records,* arrays, strings, and lists.

j. standards and conventions to be used for Ada compilation unit organization, for the use of Ada attributes for clauses and data types, and for the use of predefined Ada packages (for example, packages STANDARD and SYSTEM).

k. criteria to be used in evaluating Ada compilers to ensure they meet reliability and performance requirements (for example, compiler comparative analysis results).

1. standards and conventions to be used for Ada source code annotation (that is, *module* header information and accessibility, and source code comments and documentation conventions).

m. standards and conventions to be used for Ada source code format and presentation.

A5.3. Ada Support Environment Standards.

The requirements of the following sections are intended to state the minimum required capabilities of the *support environment* and ensure that the supplier has considered the quality of the tools which affect the software end item.

A5.3.1. Ada Compiler.

The requirements of this section are intended to ensure that the Ada compiler facilitates the generation and certification of the software end item.

a. The Ada compiler shall flag Ada pragmas, attributes, and clauses used by the software application which are not supported by the compiler, RTS, or target computer.

b. The Ada compiler shall support those features of Chapter 13 of Reference B which are used by the software application.

A5.3.2. Ada Run Time System.

For the purposes of airborne digital computer avionics systems, the Ada RTS is part of the software end item. The following apply if an Ada RTS is used.

a. The features of the Ada RTS loaded into the software end item shall be verified to the same verification type as the software application itself.

b. The supplier shall provide evidence that the Ada RTS is deterministic and meets certification requirements.

The supplier should select a compiler and RTS from a vendor who will support certification of the RTS. This means the vendor will provide data on the development and *verification* of the RTS, will perform requested testing and analyses of the RTS to support certification requirements, and will provide documentation compliant with certification requirements. The documentation should include a RTS specification, RTS implementation guidelines and standards, RTS verification and *traceability* data, and RTS source code