ONTARIO POWER GENERATION INC.

AECL Atomic Energy of Canada Limited

EACL Énergie atomique du Canada limitée

CANDU® COMPUTER SYSTEMS ENGINEERING CENTRE OF EXCELLENCE

STANDARD

CE-1003-STD
Revision 0

Standard for
Software Engineering of
Category III Software

July 2007

ONTARIO POWER GENERATION INC.

AECL Atomic Energy of Canada Limited

EACL Énergie atomique du Canada limitée

AVAILABLE

TRADEMARKS

CANDU® is a registered trademark of Atomic Energy of Canada Limited (AECL).

RELEASE SHEET

REV. NO.	ISSUE DATE	PREPARED BY AND DATE	REVIEWED BY AND DATE	APPROVED BY AND DATE	ISSUED FOR
0	July 18, 2007	G. Moum	M. Viola & J. de Grosbois	R. Hohendorf & R. Didsbury 2007 Jul 30	use

REVISION HISTORY SHEET

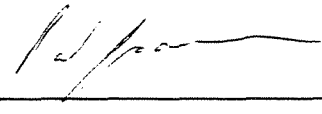
REVISION NUMBER	SECTION/PARAGRAPH	DESCRIPTION OF REVISION
0		Initial Issue. This is a reissue of the 1993 revision of this standard under the CANDU Centre of Excellence. No changes have been made.

SCI Number	
907-C-H-69002-0200	
Page 1 of 50	Rev 00
Date 1993 05	

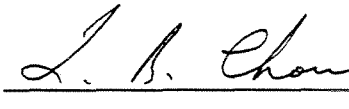
KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

Prepared by: L.F. Austin — AECL CANDU
J. Harauz — Ontario Hydro
G.J. Hinton — AECL CANDU

Approved by: 

P. Joannou
Supervising Design Engineer
Electrical and Controls Engineering Department

Issued by: 

Q.B. Chou
Manager
Electrical and Controls Engineering Department

KEYWORD: Software

SUBJECT: *Software Engineering of Category III Software*

Rev. No.	Effective Date	Prepared By	Approved By	Remarks
00	1993 05	L.F. Austin J. Harauz G.J. Hinton	P. Joannou	Issued for use

TABLE OF CONTENTS

FOREWORD 3

1.0 SCOPE AND PURPOSE 4
1.1 Purpose 4
1.2 Scope 4

2.0 OVERVIEW 5
2.1 Software Engineering Process Model 5
2.1.1 Computer System Engineering Processes 5
2.1.2 Software Development and Verification Processes 6
2.1.3 Support Processes 6
2.2 Summary of Development, Verification, and Support Processes 6
2.3 Structure of this Standard 6
2.4 Terms and Definitions 9

3.0 DEVELOPMENT PROCESSES 9
3.1 Computer System Design 9
3.2 Hardware Development and Verification 10
3.3 Software Acquisition and Qualification 10
3.4 Software Requirements Definition 10
3.5 Software Design 10
3.6 Code Implementation 10

4.0 VERIFICATION PROCESSES 11
4.1 Computer System Design Review 11
4.2 Software Requirements Review 11
4.3 Software Design Review 11
4.4 Testing 12
4.4.1 Sub-system Testing 12
4.4.2 System Integration Testing 12
4.4.3 Validation Testing 13

5.0 SUPPORT PROCESSES 13
5.1 Planning 13
5.2 Configuration Management 13

BIBLIOGRAPHY 41

GLOSSARY 21

REFERENCES FOR GLOSSARY 31

APPENDIXES

A Requirements For The Computer System Requirements Documentation
B Requirements For The Development Process Outputs
C Requirements For The Verification Process Outputs
D Requirements For The Support Outputs
E Rationale For Requirements of this Standard

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

FOREWORD

This standard has been jointly prepared by Ontario Hydro and AECL CANDU to specify the requirements for the engineering of Category III software used in real-time protective, control and monitoring systems in CANDU nuclear generating stations over the complete life of the software. This standard is the third in a family of four standards defining the engineering requirements for different categories of real-time software which are distinguished by the consequence of software failure relative to nuclear safety, non-nuclear safety, economic risk and reliability requirements. The Category III standard differs from the Category II standard primarily in the degree of formality required in design documentation and verification processes.

The software standards deliberately do not address the following CSA N286.2 generic quality assurance program requirements:

- management responsibilities and organization,
- program review and audits.

These items are addressed in the AECL CANDU and Ontario Hydro quality assurance programs.

The software standards are complemented by supplementary guidelines addressing:

- categorization of software,
- expansion on system engineering aspects in standards,
- modification of pre-developed software,
- use of configurable software,
- software procurement,
- software quality assurance.

The software standards are compliant with the CSA Q396.1.1/Q396.1.2 and ISO 9000.3 standards. They meet the SQA Manual requirements in the CSA Q396.1.1/Q396.1.2 and ISO 9000.3 standards, but in addition provide a precise definition of all software engineering processes and outputs, as well as methodology independent requirements on the outputs. They do not address contractual software developer/customer responsibilities, SQA Program policies and organizational setup, and audit requirements (similar to CSA N286.2 generic requirements not addressed as stated earlier)

A number of international and national standards have been consulted in the preparation of the software standards. The standards used appear in the bibliography. This standard also owes much of its content to the experience gained in developing, licensing, and maintaining software for the Darlington Nuclear Generating Station Shutdown System Trip Computers, Digital Control Computers, as well as other safety critical and non-safety critical software.

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

1.0 SCOPE AND PURPOSE

1.1 Purpose

This standard establishes requirements for the engineering of Category III software at CANDU nuclear generating stations. These systems normally must have high availability and highly reliable outputs, and easily modifiable functionality. The systems have minimal safety implications. The software in these systems can range in complexity from a single task executing in a single computer, to a number of cooperating tasks executing on a distributed computer system.

The software often incorporates a substantial amount of predeveloped software, including components such as operating systems, communications protocols, user interface management systems, and database managers. It is normally intended to be configured to some extent by the users.

Category III software must be reliable, reviewable, and maintainable. It is very difficult to meet these objectives for systems of the complexity described above without a clear and consistent overall computer system design and detailed information about the hardware. Therefore, this software engineering standard addresses all aspects of computer system engineering related to the software. In doing so, it provides a framework for computer system engineering, including computer system design, software requirements definition, software design, coding, interfaces with the hardware, all aspects of verification, and support activities.

For the purpose of specifying requirements, this standard defines:

(a) A minimum set of software engineering processes to be followed in creating and revising the software,

(b) The minimum set of outputs to be produced by the processes,

(c) Requirements for the content of the outputs.

These processes, outputs, and requirements for the content of the outputs are intended to cost-effectively provide a level of assurance that the delivered software will meet its design requirements and objectives. Software which meets this standard or which meets the intent of this standard with adequate justification is considered to be of "acceptable" quality with respect to the standard.

The requirements in this standard are not intended to unnecessarily constrain the methodologies and work practices used for producing the outputs. Processes and outputs may be combined or expanded as required by the project. This standard requires that the software engineering project using it, adopt and/or define and document the standards and procedures for the methodologies and work practices that will be employed. These standards and procedures shall be followed from inception until retirement of the software to provide confidence that the software is of acceptable quality.

1.2 Scope

The standard is not intended to apply directly to systems programmed in non-procedural languages such as function-block languages, although a tailored version of this standard could be used for such applications. Because this standard limits requirements on hardware engineering to areas related to the software, it is identified as a "software engineering" rather than a "computer system engineering" standard. Details about the hardware development and verification process other than what is required to engineer the software are beyond the scope of this standard. This standard does not impose any requirements on the software acquisition and qualification processes other than those requirements needed for engineering of developed software.

This standard is also not intended to be applied directly to modification of pre-existing software not developed to this standard. However, the standard does require that maintenance of software developed to this standard be a repeat of the development life cycle.

The issues not addressed in this standard are addressed in guidelines complementary to this standard (see Foreword).

The standard does not directly specify requirements for customer or user participation in the software develop-

ment process. It is stressed that user involvement is important particularly during the early project stages when identifying the system and software requirements. It is also important that software developers receive adequate training on the system to be controlled and monitored.

2.0 OVERVIEW

This section describes the process model used in this standard and describes the document's organization.

2.1 Software Engineering Process Model

The process model used in this standard is based on the premise that software engineering does not proceed in isolation from the engineering of the computer system and computer hardware. Rather it is an integral part of an overall iterative process. This interaction involves the input of system, hardware and predeveloped software requirements and design specifications to the software engineering process, as well as ongoing changes to these specifications.

The process model used in this standard is shown using two related figures, Figures 1 and 2. On these figures, unshaded processes are development processes, while shaded processes are verification processes. To keep the figures simple, feedback loops between successive processes are not shown.

The intent of the software engineering process is to achieve the required outputs through a comprehensive sequence of development and verification steps. Development involves software requirements definition followed by software design followed by code implementation. Verification of the development process outputs is performed as they are completed.

In practice, this sequential model is not followed exactly since there is feedback to preceding processes and iterations are required. However, when iteration occurs, it must be performed in conformance with the established overall process. Each release of the software throughout its use must satisfy all requirements as if the sequential model had been followed exactly.

The process model adopted in this standard requires a specific arrangement of processes, documents, and information flows. It does not, however, imply a specific sequence of activities. For example, the documents required by the standard could be prepared in strict sequence.

Alternatively, they could be developed incrementally, allowing the corresponding development processes to proceed more or less in parallel. The specific sequence of activities used to implement the process model is called the "life cycle" and under this standard must be defined in the planning process (see Appendix D).

This defined life cycle shall be followed. This standard mandates that a Software Development Plan and a Standards and Procedures Handbook shall be produced for each project, which together serve to fully define the software engineering process to be followed.

2.1.1 Computer System Engineering Processes

Figure 1 provides an overview of the computer system engineering process. To simplify this figure, details of the software development and verification processes are omitted.

The objectives of the Computer System Engineering processes are:

(a) To identify the computer system requirements.

(b) To define and verify a computer system design that meets the computer system requirements.

(c) To acquire, develop, and verify hardware and software that implements the computer system design.

(d) To integrate and test the hardware and software.

(e) To perform a validation test to confirm that the integrated system meet the requirements.

The inputs to these processes are the Computer System Requirements (CSR) documentation.

The outputs of the Computer System Engineering processes are a tested computer system and associated documentation.

2.1.2 Software Development and Verification Processes

Figure 2 provides an overview of the software development and verification processes (shown on Figure 1 as a single item).

The objectives of the Software Development and Verification processes are:

(a) To define and verify the requirements for the software

(b) To define and verify the software architecture and detailed design

(c) To develop and verify the source code.

The inputs are the Computer System Design (CSD) and the Computer Hardware Design (CHD). The output is the software, including source code and predeveloped software.

2.1.3 Support Processes

In addition to the processes shown on these two figures, there are support processes which relate to the processes shown on Figures 1 and 2. These processes are not shown on the figures in order to keep the figures simple.

The objectives of the support processes are to:

(a) Plan and fully define the development, verification, and support processes.

(b) Manage the configuration of the process outputs.

The outputs produced by these processes are used throughout the project by all other processes.

2.2 SUMMARY OF DEVELOPMENT, VERIFICATION, AND SUPPORT PROCESSES

Table 1 lists all of the processes required for Category III software, sorted by process class (development, verification, or support). For each process, the table provides a reference to the description of the process, the output(s) produced by the process, the acronym used to identify the output(s), and a reference to the minimum requirements for those outputs. The outputs shall meet detailed requirements specified in the Software Development Plan (SDP) and the Standards and Procedures Handbook (SPH).

The SDP and SPH serve to fully define the processes and sub-tier standards and procedures for the outputs, based on the minimum requirements in this standard and project-specific quality objectives. The documents mandated by this standard may reference information rather than duplicate it.

2.3 STRUCTURE OF THIS STANDARD

Sections 3, 4, and 5 describe the processes, inputs, and outputs to be produced for development, verification, and support processes, respectively. Their purpose is to aid the reader in understanding the purpose of each process and output. Sections 3, 4, and 5 do not impose any specific requirements.

Appendix A specifies minimum requirements for the Computer System Requirements.

The requirements to be met by the outputs of the development, verification, and support processes are contained in Appendices B, C, and D respectively.

Requirements to be met by any particular process are contained in Appendix D as requirements on plans, standards and procedures for that process.

A rationale for the requirements of this standard is provided in Appendix E. This appendix identifies the quality attributes which are referenced in this standard.

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

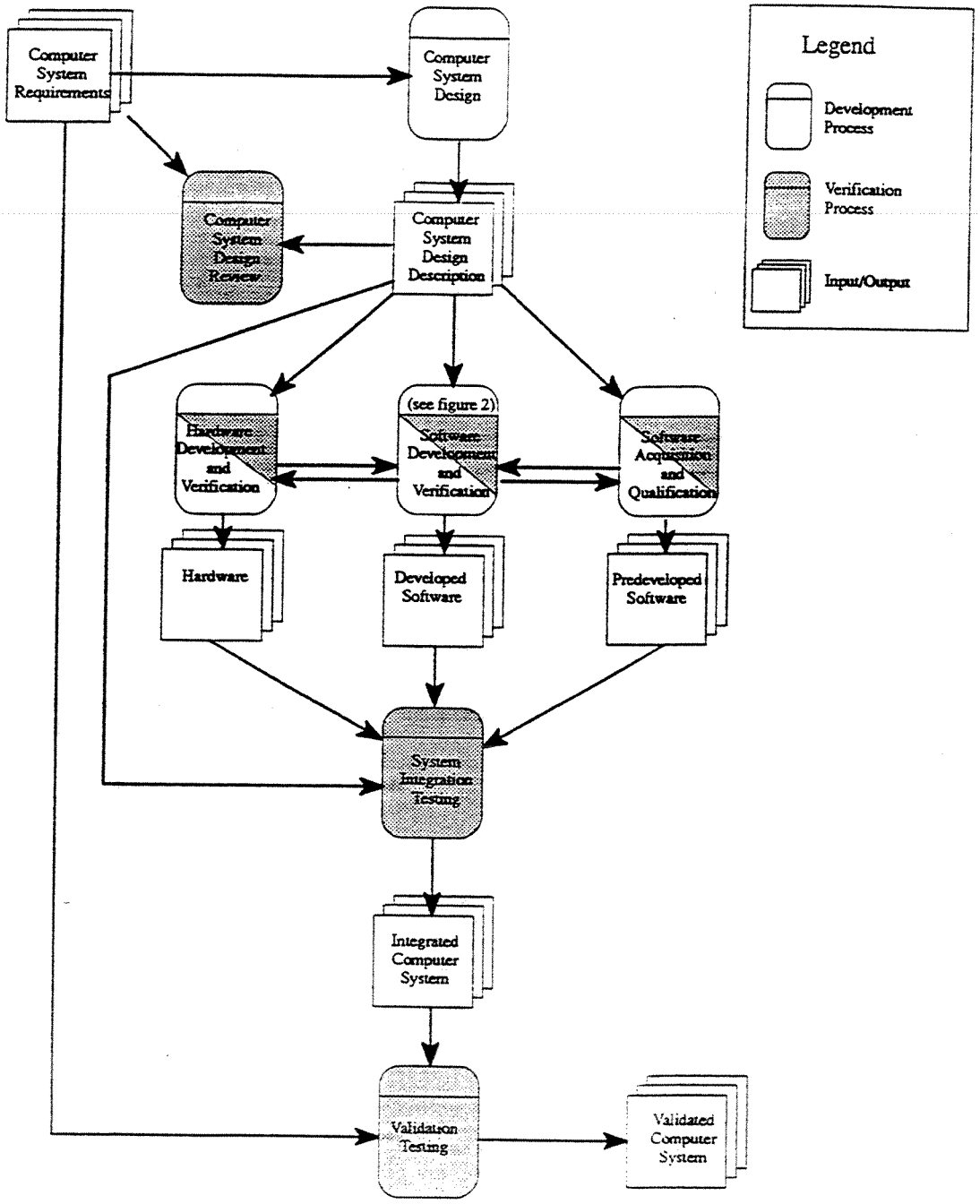


FIGURE 1
Computer System Engineering Processes

KEYWORD: Software

SUBJECT: Software Engineering for Category III Software

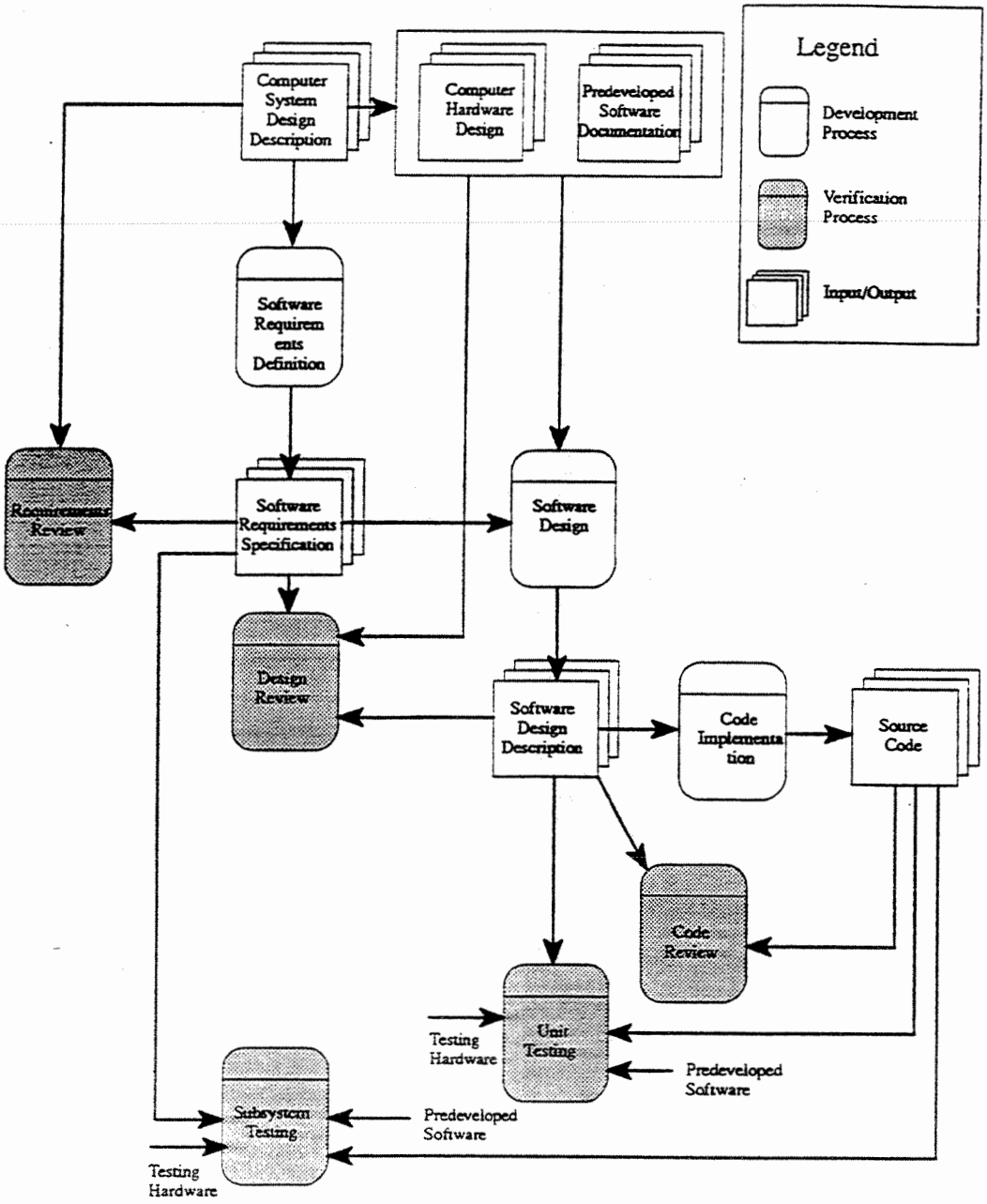


FIGURE 2
Software Development and Verification Processes

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

TABLE 1
Development, Verification and Support Processes

PROCESS	DESCRIPT.	OUTPUTS	ACRONYM	REQUIREMENTS
DEVELOPMENT				
Computer System Design	3.1	Computer System Design Description	CSD	B.1
Hardware Development and Verification	3.2	Computer Hardware Design Test Hardware	CHD –	B.2 –
Software Acquisition and Qualification	3.3	Pre-developed Software Pre-developed Software Documentation	N/A PSD	N/A –
Software Requirements Definition	3.4	Software Requirements Specification	SRS	B.3
Software Design	3.5	Software Design Description	SDD	B.4
Code Implementation	3.6	Source Code	–	B.5
VERIFICATION				
Computer System Design Review	4.1	Computer System Design Review Report	CSDRR	C.1, C.2
Software Requirements Review	4.2	Requirements Review Report	RRR	C.1, C.3
Software Design Review	4.3	Design Review Report	DRR	C.1, C.4
Sub-System Testing	4.4 4.4.1	Test Procedures Test Report	STP STR	C.5 C.8
System Integration Testing	4.4 4.4.2	System Integration Test Procedures System Integration Test Report	SITP SITR	C.6 C.8
Validation Testing	4.4 4.4.3	Validation Test Procedures Validation Test Report	VTP VTR	C.7 C.8
SUPPORT				
Planning	5.1	Software Development Plan Standards and Procedures Handbook	SDP SPH	D.1 D.2
Configuration Management	5.2	Releases Approved Change Requests Configuration Management Reports	– – –	D.1.5, D.2.5 – –

2.4 TERMS AND DEFINITIONS

The glossary contains terminology required to understand this standard.

3.0 DEVELOPMENT PROCESSES

This section describes objectives, inputs, and outputs of the development processes. These processes are:

(a) Computer System Design

(b) Hardware Development and Verification

(c) Software Acquisition and Qualification

(d) Software Requirements Definition

(e) Software Design

(f) Code Implementation

3.1 COMPUTER SYSTEM DESIGN

The objectives of the Computer System Design process are:

(a) To decompose the Computer System Requirements into functional subsystems, and define the interfaces between these functional subsystems.

(b) To define a hardware architecture, and map the functional subsystems onto the hardware architecture.

(c) To describe the major components of predeveloped software that are anticipated to be required.

The inputs to this process are the Computer System Requirements (CSR) and the Standards and Procedures Handbook (SPH). The output is the Computer System Design Description (CSD).

3.2 HARDWARE DEVELOPMENT AND VERIFICATION

The objectives of the Hardware Development and Verification process are:

(a) To define the detailed design of the hardware.

(b) To provide a testing environment closely representative of the target configuration.

The input to this process is the CSD. The outputs are the Computer Hardware Design (CHD) and hardware to be used for testing. Note that requirements for the outputs of this process are defined only with respect to software. *Other hardware engineering requirements are not defined in this standard.*

3.3 SOFTWARE ACQUISITION AND QUALIFICATION

The objectives of the Software Acquisition and Qualification process are:

(a) To acquire predeveloped software for the target system required by the CSD.

(b) To qualify this software for use in the application.

The inputs to this process are the CSD, the Software Development Plan (SDP), and SPH.

The outputs are the predeveloped software and the predeveloped software documentation. *Note that this standard does not define requirements to be met by the outputs of this process.*

3.4 SOFTWARE REQUIREMENTS DEFINITION

The objectives of the Software Requirements Definition process are:

(a) To identify and document the requirements for the software.

(b) To specify any implementation constraints required to achieve the computer system design.

The inputs to this process are the CSD and SPH. The outputs of this process could be one or more Software Requirements Specification (SRS) documents corresponding to separate functional subsystems.

3.5 SOFTWARE DESIGN

The objectives of the Software Design process are:

(a) To decompose the software requirements into modules and define the interfaces between modules.

(b) To describe each module in detail such that when all described modules are combined according to the software architecture, they satisfy all requirements specified in the SRS.

(c) To produce a detailed software design which is compatible with the detailed computer hardware design described in the CHD.

(d) To select a computer language or languages and translator(s) to be used for code development.

The inputs to this process are the SRS(s), the SPH, the CHD, and the predeveloped software documentation (PSD). The output can be one or more Software Design Descriptions (SDDs).

3.6 CODE IMPLEMENTATION

The objectives of the Code Implementation process are:

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

(a) To translate the SDD into source code,

(b) To translate the source code into executable code,

(c) To integrate the executable code,

(d) To debug the executable code,

(e) To create all required databases complete with appropriate initial data.

The inputs to this process are the SDD(s) and the SPH. The output is the source code.

The source code is a translation of the design as specified in the SDD(s).

4.0 VERIFICATION PROCESSES

This section describes the objectives, inputs, and outputs of the verification processes. These processes are listed below:

(a) Computer System Design Review

(b) Software Requirements Review

(c) Software Design Review

(d) Subsystem Testing

(e) System Integration Testing

(f) Validation Testing

4.1 COMPUTER SYSTEM DESIGN REVIEW

The objectives of the Computer System Design Review process are:

(a) To verify that the Computer System Design (CSD) meets the intent of the requirements specified in the Computer System Requirements (CSR), and that the design decisions are consistent with good computer system engineering practice,

(b) To identify ambiguities and incompleteness in the requirements specified in the CSR,

(c) To verify that the CSD addresses all requirements specified in the CSR,

(d) To check the justification for inclusion of any functionality outside the scope of the requirements to confirm that the resulting design is consistent with the intent of the requirements,

(e) To verify that the CSD meets the requirements in the Standards and Procedures Handbook (SPH).

The inputs to this process are the CSD, CSR, and SPH. The output is the Computer System Design Review Report (CSDRR).

4.2 SOFTWARE REQUIREMENTS REVIEW

The objectives of the Software Requirements Review process are:

(a) To verify that the requirements specified in the Software Requirements Specification (SRS) are complete, adequate, and consistent with the intended application,

(b) To identify ambiguities and incompleteness in the requirements specified in the SRS,

(c) To verify that the SRS meets the requirements of the CSD,

(d) To identify ambiguities and incompleteness in the requirements specified in the CSD,

(e) To verify the justification for including any requirements and design constraints in the SRS which were not derived from the CSD,

(f) To verify that the SRS meets the requirements in the SPH.

The inputs to this process are the SRS, SPH and CSD. The output is the Requirements Review Report (RRR).

4.3 SOFTWARE DESIGN REVIEW

The objectives of the Software Design Review process are:

(a) To verify that the design decisions are consistent with good software engineering practice, i.e. that the defined software architecture is the best one selected,

(b) To verify that the Software Design Description(s) (SDD) meets the intent of the requirements specified in the SRS,

(c) To check the justification for inclusion of any functionality outside the scope of the requirements to confirm that the resulting design is consistent with the intent of the requirements,

(d) To verify that the SDD meets the requirements in the SPH.

The inputs to this process are the SDD, SPH, SRS, Computer Hardware Design (CHD), and Predeveloped Software Documentation (PSD). The output is the Design Review Report (DRR).

4.4 TESTING

Each testing process includes:

(a) Preparing procedures to specify the environment in which the code is tested, detailing the steps to be followed by the tester, and specifying the set of inputs, execution conditions, and expected results.

(b) Verifying the test procedure with the objective to ensure that:

(1) the required test coverage is provided,

(2) the procedures are specified so tests can be repeated,

(3) the tests are prepared as specified in the SPH.

(c) Implementing the test procedure, recording the outputs, and assessing the test results.

(d) Verifying the test report with the objective to ensure that:

(1) tests were executed as specified in the test procedures,

(2) test results were recorded and analyzed,

(3) test reports meet the requirements in the SPH.

The outputs from this process are test reports and test procedures. All of these tests are preformed on test hardware that is representative of the target environment. Different levels of testing may require different amounts of hardware to represent the target.

4.4.1 Subsystem Testing

The objectives of the Subsystem Testing process are:

(a) To translate the source code implementing each of the functional subsystems into executable code without errors,

(b) To test that the software implementing each of the functional subsystems meets the requirements specified in the corresponding SRS,

(c) To find any errors that may be present in the software, hardware, and predeveloped software interfaces,

(d) To find any errors that may be due to stress conditions and timing problems,

(e) To find any errors that may be present when performing fail-safe functions, detecting error conditions, and performing error recovery.

The inputs to this process are the SRS, SPH, source code, predeveloped software, and test hardware. The outputs are the Subsystem Test Procedures (STP), software and hardware required to simulate the environment, and the Subsystem Test Report (STR).

4.4.2 System Integration Testing

The objectives of the System Integration Testing process are:

(a) To combine the software functional subsystems and hardware in a systematic manner to generate a compatible system,

(b) To test that the interfaces between functional subsystems behave as specified in the CSD,

(c) To test that the physical interfaces between computers behave as specified in the CSD,

(d) To test that the functional subsystems perform in an integrated environment as specified in the CSD,

(e) To test that the system architectural features, including fault tolerance features, perform as specified in the CSD.

The inputs to this process are the CSD, SPH, developed software, predeveloped software and test hardware. The outputs are the System Integration Testing Procedures (SITP), the System Integration Test Report (SITR), and the integrated computer system.

4.4.3 Validation Testing

The objective of the Validation Testing process is to test that the integrated computer system meets the requirements specified in the CSR.

The inputs to this process are the CSR, SPH, and integrated computer system. The outputs are the Validation Test Procedures (VTP), the Validation Test Report (VTR), and the validated computer system.

5.0 SUPPORT PROCESSES

This section describes the objectives, inputs and outputs of the support processes. These processes are listed below:

(a) Planning

(b) Configuration Management

5.1 Planning

The objectives of the Planning process are:

(a) To define all processes in the software lifecycle (as listed in Table 1) for the software engineering project relating to how and when they are to be done and who is to do them at the start of the project.

A specific model for the software lifecycle is adopted as a focus for planning and for the activities mapped to it,

(b) To define verification requirements, training requirements and user participation in the project,

(c) To identify and document all project-specific standards and procedures,

(d) To define an organization that meets the independence requirements (defined in Appendix D.1.1) for personnel participating in the various software engineering processes and that ensures that work is carried out objectively and effectively,

(e) To establish estimates, schedules, budgets, and resource requirements. These include the effort required to develop procedures, support tools, and facilities.

The inputs to this process are the Computer System Requirements (CSR), this standard, and exiting sub-tier standards and procedures. The outputs are the Software Development Plan (SDP) and the Standards and Procedures Handbook (SPH).

5.2 Configuration Management

The objectives of the Configuration Management process are:

(a) To identify the software configuration at discrete points in time to ensure that the correct version of each process output is being used at any point in time,

(b) To control all changes made to the software,

(c) To ensure that changed configuration items are developed and verified with the same rigour as applied to the original items,

(d) To define under which circumstances verification processes (such as technical reviews) must be carried out on revisions to the development outputs,

(e) To provide an on-going analysis of encountered errors to be used as input for continuous improvements to the sub-tier standards and procedures defined in the SPH.

KEYWORD: Software

SUBJECT: Software Engineering for Category III Software

The inputs to this process include all items whose configuration is managed (this includes, as a minimum, all project documents listed in Table 1 including the SDP and SPH) and all software change requests.

The outputs of this process include the controlled version of all project documents (at least all of the outputs listed in Table 1), software releases, approved change requests, and software configuration status reports.

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

BIBLIOGRAPHY

AECL CANDU, "CANDU 3 Software Quality Assurance Manual," 74-01913-MAN-003, Rev. 0, April, 1991.

AECL CANDU, "Safety Software Design Principles," 74-66300-DG-001.

Babich, W.A., "Software Configuration Management", Addison-Wesley Publishing Company, 1986.

Bishop, P.G (ed), "Dependability of Critical Computer Systems 3", Guidelines produced by EWICS TC7, 1, Elsevier Applied Science, London, 1990.

Booch, G., "Software Engineering with Ada", Benjamin /Cummings Publishing Company, Menlo Park CA, 1987.

Boehm, B.W., "Software Engineering Economics", Prentice-Hall Inc., 1981.

Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., Macleod, G.J., Merritt, M.J., "Characteristics of Software Quality", TRW Series of Software Technology, Volume 1, TRW and North-Holland Publishing Company, 1978.

Buhr, R.J.A, "System Design with Ada", Prentice-Hall, Englewood Cliffs, NJ, 1984.

CAN/CSA-Q396.1.1-89 "Quality Assurance Program for the Development of Software Used in Critical Applications".

CAN3-N286.2-86 "Design Quality Assurance for Nuclear Power Plants".

Daughtrey, H.T., "Measuring the Full Range of Software Qualities", Babcock & Wilcox, ASQC Quality Congress, Dallas, 1988.

DOD-STD-2167a, "Defense System Software Development", February 1988, and the following data item descriptions:

- DI-CMAN-80534, "System/Segment Design Document (SSDD)"
- DI-MCCR-80030, "Software Development Plan (SDP)"
- DI-MCCR-80025, "Software Requirements Specification (SRS)"
- DI-MCCR-80026, "Interface Requirements Specification (IRS)"
- DI-MCCR-80027, "Interface Design Document (IDD)"
- DI-MCCR-80012, "Software Design Document (SDD)"
- DI-MCCR-80029, "Software Product Specification (SPS)"
- DI-MCCR-80013, "Version Description Document (VDD)"
- DI-MCCR-80014, "Software Test Plan (STP)"
- DI-MCCR-80015, "Software Test Description (STD)"

Dorfman, M., Thayer, R.H., "Standards, Guidelines, and Examples on System and Software Requirements Engineering", Glossary, IEEE Computer Society Press, 1990.

European Space Agency, "ESA Software Engineering Standards," ESA PSS-05-0 Issue 2, 1991 February.

Hicks, D.B., Harauz, J. and Hohendorf, R.J., "Software Quality Assessment Process", CANDU Owner's Group Conference, Toronto, 1990 November.

IEC 65A (Secretariat) 94, Draft, "Software for Computers in the Application of Industrial Safety Related Systems".

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

BIBLIOGRAPHY (Continued)

IEC 880 "Software for Computers in the Safety Systems of Nuclear Power Stations".

IEEE P1062/021, IEEE Draft Standard for a Software Quality Metrics Methodology, April 1, 1990.

IEEE P1074/04, "Draft Standard for Software Life Cycle Processes", 1989.

IEEE Std 830-1984 "IEEE Guide to Software Requirements Specification".

IEEE Std 730-1981 (R 1984) "IEEE Standard for Software Quality Assurance Plan".

IEEE Std 982.2-1988 "IEEE Guide for the Use of IEEE Standard Dictionary".

"IEEE Transactions on Software Engineering, Special Section on Software Engineering Project Management", Vol SE-10, No. 1, January, 1984.

ISO - Committe Draft ISO Standard for Software Life-Cycle Process, JTC1(ISO/IEC)-SC7, August 30, 1991.

ISO/IEC JTC1/SC7/WG3, DP 9126, "Software Product Evaluation - Quality Characteristics and Guidelines for Their Use", 1990.

ISO/IEC JTC1/SC7/WG3, N-136, "Evaluation of Methods and Tools", 1989.

ISO/IEC JTC1/SC7/WG3/SG2, "Foundations of Software Quality Assessment", R.E. Nance and J.D. Arthur, Systems research Center and The Department of Computer Science, Virginia Tech, 1990.

Jensen, R.W., Tonies, C.C., "Software Engineering", Prentice-Hall Inc., 1979.

Juran, J.M., Gryna, F.M., (ed), "Juran's Quality Control Handbook", 4th edition, McGraw Hill Book Company, 1988.

Kopetz, H., "Software Reliability", Macmillan Press Ltd., 1979.

Lamb, D.A., "Software Engineering, Planning for Change", Englewood Cliffs, NJ: Prentice-Hall, 1988.

Martin, J., McClure, C., "Software Maintenance", Prentice-Hall Inc., 1983.

MOD Interim Defence Standard 00-55 (Draft), "Requirements for the Procurement of Safety Critical Software in Defence Equipment".

MOD Interim Defence Standard 00-56 (Draft), "Requirements for the Analysis of Safety Critical Software".

Myers, G.J., "The Art of Software Testing" New York, N.Y. Wiley-Interscience, July, 1979.

NSD Guideline for Categorization of Software in Ontario Hydro's Nuclear Facilities with Respect to Nuclear Safety.

NUREG/CR-4640, PNL-5787, "Handbook of Software Quality Assurance Techniques Application to the Nuclear Industry".

Ontario Hydro, Darlington Engineering Department, "Darlington Nuclear Generating Station A Control Computers Software Quality Assurance Manual".

Ontario Hydro, "Quality Engineering Manual".

Ontario Hydro Review of Pickering NGS A Power House Emergency Venting System - Random Test Specification, Internal Report, July 3, 1990

Ould, M.A., Unwin, C. (ed), "Testing in Software Development", British Computer Society Monograph, Cambridge University Press, 1988.

Parnas, D.L., "Proposed Standard for Software for Computers in the Safety Systems of Nuclear Power Stations (based on IEC Standard 880)" (DLP-880), March 1991.

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

BIBLIOGRAPHY (Continued)

Parnas, D.L., Madey, J., "Functional Documentation for Computer Systems Engineering", Kingston, Ontario: Telecommunications Research Institute of Ontario, Queen's University, Technical Report 90-287, ISSN O836-0227, September 1990.

Pressman, R.S., "Software Engineering", New York, McGraw-Hill Book Company, 1987.

Rannem, S., and E. Hung, "On Factors Contributing to Quality of Nuclear Control Computer Software", Ontario Hydro.

Redmill, F.J. (ed), "Dependability of Critical Computer Systems 1", Guidelines produced by EWICS TC7, 1, Elsevier Applied Science, London, 1988.

Redmill, F.J. (ed), "Dependability of Critical Computer Systems 2", Guidelines produced by EWICS TC7, 1, Elsevier Applied Science, London, 1989.

Rook, P. (ed) "Software Reliability Handbook", Elsavier Applied Science, London, 1990.

Smith, D.J., Wood, K.B., "Engineering Quality Software", 2nd edition, London: Elsevier Applied Science, 1989.

Tausworthe, R.C., "Standardized Development of Computer Software", Prentice-Hall Inc., 1977.

AW 366653
Page is left blank intentionally Apr. 30/08

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

GLOSSARY

This glossary defines any terminology which is not explicitly defined within this standard.

The definitions used in this glossary come from a variety of sources. Those definitions which are either taken directly or paraphrased from a source reference that source in brackets "[...]" immediately following the definition. The use of a source does not imply an exact quote from it, but rather, acknowledges the originating definition. A list of the references used in this glossary appears at the end of this section.

acceptable quality–All of the requirements which this standard imposes are met.

animation–A software requirement analysis tool that refers to the process of being able to walk through a specific portion of the specification by using certain inputs to determine its behaviour under different circumstances. [IEEE]

assembler–Tool for translating assembly language code into a form which is executable on a computer.

assembly language–A representation of computer instructions and data that usually has a one-to-one correspondence with machine language. Assembly language is more representative of the application program than machine language. [IEEE]

baseline–A set of software items that has been formally reviewed and agreed upon; it then serves as the basis for further development, and can be changed only through change control procedures. [IEEE]

change control–The process by which a change is proposed, evaluated, approved or rejected, scheduled, and tracked. [IEEE]

change request–A formal written request that a modification or correction be made to system requirements, software requirements, software design, or code.

CHD–Computer Hardware Design

code–Any or all of source code, executable code, and any other files produced during translation of source into executable (e.g., object files, intermediate files, listings, and maps).

code implementation–The phase of software engineering which involves the conversion of a software design into code and executable code. [IEEE]

cohesion–The degree to which the tasks performed by a single program are functionally related. Cohesion is a measure of the strength of association of the elements within a program. [IEEE]

compiler–Tool for translating computer language source code into a form which is executable on a computer system.

completeness–Quality attribute which refers to the extent to which all of the software's required functions and design constraints are present and fully developed in the SRS, SDD, and code.

computer–A functional programmable unit that consists of one or more associated processing units and peripheral equipment that is controlled by internally stored executable code and that can perform substantial computation including numerous arithmetic operations or logic operations, without human intervention. [IEEE]

computer system–A system composed of computer(s), peripheral equipment such as disks, printers and terminals, and the software necessary to make them operate together. [IEEE]

configuration–The way in which configuration items are linked at a point in time. [CAN/CSA-Q396.1.1-89]

configuration control–The process of controlling the changes and maintaining the integrity of a configuration. [CAN/CSA-Q396.1.2-89]

configuration item–An aggregation of software, or any of its discrete portions, that satisfies an end-use function and is designated for configuration management. [IEEE]

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

GLOSSARY (Continued)

configuration management–The process of identifying configuration items , controlling changes, and maintaining the integrity and traceability of the configuration. [CAN/CSA-Q396.1.1-89]

consistency–Quality attribute which refers to the extent to which the CSD, SRS, SDD, and code contain uniform notations, terminology, comments, symbology, and implementation techniques.

context–The information environment of a system. A context includes sources of information and sinks (destinations) for information. A system's context is often described using a context diagram, which illustrates the system, the sources, the sinks, and the interfaces between them. See also context boundary, context diagram.

context boundary–The demarcation between the system and the system context. The inputs and outputs of a system are defined with respect to the context boundary.

context diagram–The top-level diagram of a levelled data flow diagram set. It portrays all the net inputs and outputs of the system, but shows no decomposition. [DeMarco]

correctness–Quality attribute which refers to the ability of the CSD, SRS, SDD, and code to describe or produce the specified outputs when given the specified inputs, and the extent to which they match or satisfy the requirements in the CSR, DID.

coupling–A measure of the interdependence among programs or modules. Coupling is the amount of information shared between two programs or modules.

coverage matrix–A method of indicating which portion of an output document satisfies the requirements of an input document to assist traceability and completeness checks.

CSD–Computer System Design.

CSDRR–Computer System Design Review Report.

CSR–Computer System Requirements.

data flow–the sequence in which data transfer, use, and transformation are performed during the execution of a computer program. [IEEE]

data flow diagram (DFD)–A diagram that depicts data sources, sinks, data storage, and processes performed on data as nodes, and logical flow of data as links between the nodes. [IEEE]

data structure–A representation of the logical relationship among individual data elements. [PRESSMAN]

database–A large, organized collection of information that is accessed via software. [PRESSMAN]

debugging–The activity of detecting errors, then determining the exact nature and location of a code error and either fixing or repairing the error. [MYERS]

deliverable–Contracted software item to be delivered by the developer. [CAN/CSA-Q396.1.1-89]

design constraint–A requirement that impacts or constrains the design of software. Examples of design constraints are physical requirements, performance requirements, software development standards, and software quality assurance standards. [IEEE]

design decision–The selection from a collection of alternate design solutions based on analysis.

design notes–A means of recording the rationale, experience, options, and trade-offs behind design decisions.

DRR–Design Review Report.

entity-relationship diagram (ERD)–A diagram that depicts a set of real-world entities and the logical relationships between them. [IEEE]

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

GLOSSARY (Continued)

entry point–The address in a program at which it will begin execution.

equivalence partitioning–A testing technique which relies on looking at the set of valid inputs specified for a program–its domain–and dividing it up into classes of data that should, according to the specification, be treated identically. These equivalence classes will not overlap and one set of test data is then chosen to represent each equivalence class. [TESTING]

evolutionary development model–A software development process whose stages consist of expanding increments of an operational software product. The directions of evolution are determined by operational experience. The evolutionary developmental model gives users a rapid initial operational capability and provides a realistic operational basis for determining subsequent product improvement. [IEEE]

executable code–A program in a language that can be directly executed by a computer. [IEEE]

exit point–The address in a program at which it can terminate execution.

failure mode–A way in which the system can fail to meet its requirements.

fault–A software defect that can lead to a failure. [MOD 00-55]

fault tolerance–The built-in capability to provide continued correct execution, such as the provision of service as specified, in the presence of a limited number of hardware or software faults.

feasibility study–A study to determine the benefits and costs, in terms of time, money, and personnel required, of a proposal to accomplish a given purpose, usually with a recommendation or proposed alternatives. [IEEE]

functional requirement–The statements that define the essential features of the software components, along with the technical constraints and conditions that are to be met. [CAN/CSA-Q396.1.1-89]

functional subsystem–A logical component of a computer system. A functional subsystem performs a subset of the functions performed by the computer system. A functional subsystem has logical interfaces to other functional subsystems and to the computer system context. It is defined by computer system designers to organize functional requirements into related subsets, and to ensure that the relation between those requirements is understood.

For a CANDU Digital Control Computer (DCC), an example of a functional subsystem might be the Reactor Regulating System (RRS), or the channel power mapping subsystem. The RRS might have as logical inputs the demand power and a channel power map, and as logical outputs a set of zone reactivity setpoints.

For a plant display system, a functional subsystem might be the historical data logging subsystem, or the plant status display subsystem. The historical data logging subsystem might have as logical inputs the current plant status information, and as logical outputs a series of trends, extreme values, and averages.

graceful degradation–Stepwise reduction of functions in response to detected failures while essential functions are maintained. [IEC 65A(Secretariat)94]

guidelines–Sets of well-documented standards and definitions that are used to guide an activity or task. Guidelines are usually not rules or procedures but allow judgement and flexibility. [IEEE]

hardware configuration–An arrangement of hardware components.

hold points–Those points in the development project beyond which the work can not proceed without approval of specific documents.

information hiding–A software development technique in which each module's interface reveals as little as possible about the module's inner workings. Other modules are forbidden to use information about the module that is not in its interface specification. [IEEE]

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

GLOSSARY (Continued)

information hiding principle–A heuristic used for decomposing systems that states that decisions that are expected to change together should be kept together. Each module should be designed to 'hide' the decision from the others. See also: information hiding technique. [Parnas-decomp]

information hiding technique–A software development technique in which each module's interface reveals as little as possible about the module's inner workings. Other modules are forbidden to use information about the module that is not in its interface specification. The technique is a specific way to implement the information hiding principle. [IEEE]

inspection–A semiformal to formal evaluation technique in which software requirements, design, or code are examined in detail by a person or group other than the originator to detect faults, violations of development standards, and other problems. The review members are peers (equals) of the developer. Traditional error data is collected during inspections for later analysis and to assist in future inspections. Sometimes called a walk through or peer review. [IEEE]

integrated computer system–The combination of hardware and software that implements the computer system design (CSD).

interface:

(1) A shared boundary. An interface might be a hardware component to link two devices or it might be a portion of storage or registers accessed by two or more programs.

(2) To interact or communicate with another subsystem component or organizational entity. [IEEE]

inter-task communication–a transfer of information between tasks, such as synchronization information or data. This is usually performed using an inter-task communications service.

inter-task communications service–A mechanism for performing inter-task communications. Typical mechanisms for synchronization information include: semaphores, signals, traps, and interrupts. Typical mechanisms for data communications include: message queues, pipes, shared memory, and mailboxes.

lifecycle–All the steps or phases an item passes through during its useful life. [IEEE]

logical–Implementation-independent. [ESA]

logical input–a logical interface that provides information to a logical item (such as to a functional subsystem).

logical interface–An implementation-independent interface between functional subsystems, or between a functional subsystem and the system context. Logical interfaces are identified in the functional decomposition section of the CSD and used by the SRS. A logical interface usually represents a value occurring in the environment (e.g., a flux value in the reactor core) or a value derived from one of these. Contrast: physical interface. [ESA]

logical model–An implementation-independent representation of a real-world process. Contrast: physical model. [ESA]

logical output–a logical interface that receives information from a logical item (such as from a functional subsystem).

machine language–The instructions that are directly executable by a computer. [IEEE]

methodology–A general approach to solving an engineering problem. [IEEE]

milestone–A scheduled event that is used to measure progress. [IEEE]

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

GLOSSARY (Continued)

modifiability–Quality attribute which refers to the characteristics of the CSD, SRS, SDD, and code which facilitate the incorporation of changes.

modularity–Quality attribute which refers to the extent to which the SDD is composed of discrete components such that a change to one component has minimal impact on the others.

module–A collection of data structures and programs which can act on those data structures. The data structures can only be accessed externally through access programs provided by the module.

naming convention–A guideline for the naming of various entities within a software system.

non-deliverable–A document or software item which, although useful for development, is not required to be delivered by the developer.

notation–A physical, graphical, or textual means of describing a software requirement, design, or code. [IEEE]

operating system–Software which provides application processes with controlled access to computer resources. As a minimum, it provides orderly system startup and termination functions and exception handling capability.

physical–Implementation-dependent. [ESA]

physical input–a physical interface that provides information to a physical entity (such as to a software task or to a computer.)

physical interface–An implementation-dependent interface. Physical interfaces between computers in the computer system are identified in the hardware architecture section of the CSD, and are described in detail in the CHD. Physical interfaces between software components (e.g. between tasks or modules) and between the software and the computer hardware are described in the SDD. The SDD also provides a mapping between logical interfaces described in the SRS and physical interfaces used in the software.

A physical interface between pieces of hardware is usually implemented with some kind of medium (e.g. a wire) and a communications protocol. A physical interface between tasks is implemented using inter-task communications. A physical interface between modules is implemented using procedure calls or shared variables.

A physical interface between software and computer hardware is usually implemented with an I/O device that provides either programmed I/O (input and output registers) or memory-mapped I/O. Often hardware interrupts are used as part of the signalling protocol used to control physical interfaces between software and hardware. Contrast: logical interface. [ESA]

physical model–An implementation-dependent representation of a real-world process. Contrast: logical model. [ESA]

physical output–a physical interface that receives information from a physical entity (such as from a software task or from a computer.)

performance requirement–A requirement specifying a performance characteristic; for example, speed, accuracy, and frequency. [IEEE]

predeveloped software–Software which has been produced prior to the issuing of a contract or purchase order, or to satisfy a general market need. [CAN/CSA-Q396.1.1-89]

predictability–Quality attribute which refers to the extent to which the functionality and performance of the software are deterministic for a specified set of inputs.

procedure–A written set of rules to be followed in the performance of a task.

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

GLOSSARY (Continued)

procedural language–A language in which the user specifies a set of executable operations and the sequence in which they are to be performed, i.e., Fortran, assembler, Pascal.

program–A uniquely identifiable sequence of instructions and data which is part of a module (e.g., main program, subroutine, and macro).

project–An activity characterized by a start date, specific objectives and constraints, established responsibilities, a budget and schedule, and a completion date. (If the objective of the project is to develop software, then it is sometimes called a software development or software engineering project). [IEEE]

prototyping:

(1) A software requirements analysis strategy that includes the idea of building a system prototype to give early visibility of its actual functioning and operational behaviour to the designer and the customers.

(2) The development of a "quick and dirty" system, beginning with the very minimum of specifications that have been prepared. The purpose is to show the users what they are asking for and to give them a working knowledge of the result that can be achieved. [IEEE]

qualification–The review and testing of a product to ensure that it meets stated requirements.

quality–The totality of features and characteristics of a product or service that bears on its ability to satisfy given needs. [IEEE]

quality attributes–The features and characteristics of a software component that determine its ability to satisfy requirements. [CAN/CSA-Q396.1.1-89]

The quality attributes relevant for the standard are defined in the Quality Attributes section.

real-time–Pertaining to a system or mode of operation in which computation is performed during the actual time that an external process occurs, in order that the computation results can be used to control, monitor, or respond in a timely manner to the external processes. [IEEE]

record–A set of related items treated as a unit. For example, in stock control, the data for each invoice could constitute one record. [IEEE]

redundancy–In fault tolerance, the presence of auxiliary components in a system to perform the same or similar functions as other elements for the purpose of preventing or recovering from failures. [IEEE]

regression testing– The rerunning of test cases that a program has previously executed correctly in order to detect errors created during software correction or modification activities. [IEEE]

release–A planned output consisting of a verified and validated version of the software.

reliability requirements–A requirement specifying the probability of not encountering a sequence of inputs which lead to a failure. [PARNAS, *et al.*]

requirement:

(1) A capability needed by a user to solve a problem or to achieve an objective.

(2) A capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.

(3) The set of all requirements that form the basis for subsequent development of the software or software component. [IEEE]

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

GLOSSARY (Continued)

review:

(1) A formal meeting at which a product or document is presented to interested parties for comment and approval. It can be a review of the management and technical progress of the development project.

(2) The formal review of an existing or proposed design for the purpose of detection and remedy of design deficiencies that could affect fitness for use and environmental aspects of the product, process, or service, and for identification of potential improvements in performance, safety or economy. [IEEE]

robustness–Quality attribute which refers to the extent to which the CSD, SRS, SDD, and code require and implement the ability to continue to perform despite some subsystem failure. See quality attributes.

RRR–Requirements Review Report.

safety critical software–Software which is part of a special safety system and which is required for the special safety system to meet its minimum allowable performance standards.

safety-related software–Software which is part of a safety-related system and which is required for the safety-related system to meet its minimum allowable performance standards.

safety requirements–The requirements which are concerned with the prevention or mitigation of the effects of failures which could lead to an unsafe system state.

SDD –Software Design Description.

SDP –Software Development Plan.

security-Fault tolerance against deliberate interaction faults (intrusion) from internal or external sources. [IEC 65A (Secretariat) 94]

self-modifying code–Code which, by design, achieves its function by overwriting itself.

semantics–A branch of linguistics that is concerned with the nature, structure, development, and changes in the meaning of words. Semantics is the meaning of a sentence. [IEEE]

simulation modelling:

(1) Physical or mathematical representation of a system to expedite the evaluation of selected system parameters. Simulation is used to explore the effects that alternative system characteristics will have on system performance without actually producing and testing each alternative system.

(2) Use of an executable model to represent the behaviour of an object. During testing, the computational hardware, the external environment, and even code segments may be simulated. [IEEE]

skills inventories–A list of personnel available for a project with a list of each person's skills which are pertinent to the project.

software–A set of programs, associated data, procedures, rules, documentation, and materials concerned with the development, use, operation, and maintenance of a computer system. [CAN/CSA-Q396.1.1-89]

software component–A software item that is placed under configuration control. [CAN/CSA-Q396.1.1-89]

software engineering:

(1) The practical application of computer science, management, and other sciences to the analysis, design, construction, and maintenance of software and associated documentation.

(2) An engineering science that applies the concept of analysis, design, coding, testing,

GLOSSARY (Continued)

documentation, and management to successful completion of large custom-built software.

(3) The systematic application of methods, tools, and techniques to achieve a stated requirement or objective for an effective and efficient software system.

(4) The application of scientific principles to:

- the early transformation of a problem into a working software solution, and
- subsequent maintenance of that software until the end of its useful life. [IEEE]

software item–A functionally or logically distinct part of the software. [CAN/CSA-Q396.1.1-89]

source code–Computer instructions and data definition statements expressed in a form suitable for input to a compiler, assembler, other translator, or interpreter. [IEEE]

SPH –Standards and Procedures Handbook.

SRS –Software Requirements Specification.

standard:

(1) A standard is an approved, documented, and available set of criteria used to determine the adequacy of an action or object.

(2) A document that sets forth the standards and procedures to be followed on a given project or by a given organization.

(3) A software engineering standard is a set of

- procedures that define the processes for and
- descriptions that define the quantity and quality of a product from a software engineering project. [IEEE]

state transition diagram–A diagram that depicts the states that a system or component can assume, and shows the events or circumstances that cause or result from a change from one state to another. Also known as "state diagram." [IEEE]

SITP –System Integration Test Procedures.

SITR –System Integration Test Report.

STP –Subsystem Test Procedures.

STR –Subsystem Test Report.

structured design–

(1) Any disciplined approach to software design that adheres to specified rules based on principles such as modularity, top-down design, and stepwise refinement of data, system structures, and processing steps.

(2) The result of applying the approach in (1). [IEEE]

structured English–In software engineering (structured analysis methodology), a subset of the English language with limited syntax, limited vocabulary, and an indentation convention to call attention to logical blocking: a meta-language for process specification.

Structured English uses imperative English verbs without adjectives and adverbs, terms from the data dictionary, and reserved words to denote logic. Its "programming language" syntax consists of simple sentences, closed-end decisions, closed-end repetition, and combinations of these three. [IEEE]

structured programming–A programming technique in which programs are constructed of a basic set of control structures, each having one entry and one exit. The set of control structures typically includes: sequence of two or more instructions, conditional selection of one of two or more sequences of instructions, and repetition of a sequence of instructions. [IEEE]

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

GLOSSARY (Continued)

structuredness–Quality attribute which refers to the extent to which the CSD, SDD and code possess a definite pattern in their interdependent parts. This implies that the design has proceeded in an orderly and systematic manner (e.g., top-down design), has minimized coupling between modules, and that standard control structures have been followed during coding resulting in well structured software.

subsystem–A group of assemblies or components or both combined to perform a single function. [IEEE]

syntax–A branch of linguistics that is concerned with the arrangement of words or elements in a sentence and their relationships with each other. Syntax is the relationships of word groups, phrases, clauses, and sentences, that is, sentence structure. [IEEE]

system–A collection of hardware, software, people, facilities, and procedures organized to accomplish some common objectives. [IEEE]

system engineering–The application of scientific and engineering efforts to:

(1) transform an operational need into a description of system performance parameters and a system configuration through the use of an iterative process of definition, synthesis, analysis, design, testing, and evaluation;

(2) integrate related technical parameters and ensure compatibility of all related, functional, and program interfaces in a manner that optimizes the total system definition and design. [IEEE]

target–The computer hardware to be used in actual operation of the computer system.

task :

(1) A sequence of instructions treated as a basic unit of work by the supervisory program of an operating system.

(2) In software design, a software component that can operate in parallel with other software components. [IEEE]

technical review–see "review," definition (1).

test–A set of one or more test cases. [IEEE]

test case–A specific set of test data and associated procedures developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. [IEEE]

test procedure–A document specifying a sequence of actions for the execution of a test. [IEEE]

testing–The process of detecting software errors in order to produce software that meets requirements under all credible operating conditions.

top-down design–The process of designing a product by identifying its major components, decomposing them into their low-level components, and iterating this process until the desired level of detail is achieved. [IEEE]

translator–Tool for translating from a higher level language to a lower level language, e.g., a compiler or assembler.

understandability–Quality attribute which refers to the extent to which the meaning of the CSD, SRS, SDD, and code are clear to the reader.

unit–a separately compilable component of the source code. Usually corresponds to a module.

unused code–A portion or portions of software which is not intentionally executed and which serves no purpose in satisfying the software requirements or design.

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

GLOSSARY (Continued)

user–A consumer of the service provided by a system. Distinguish between customer (e.g. a company) and the end user (e.g., an employee).

UTP –Unit Test Procedures

UTR –Unit Test Report.

validated computer system–The combination of hardware and software that implements the computer system requirements (CSR), and has completed validation testing.

validation–The process of determining the correctness of the final product with respect to the initial requirements. [IEEE]

verifiability–Quality attribute which refers to the extent to which the CSD, SRS, SDD, and code have been written to facilitate verification using both static methods and testing.

verification–The process of determining whether the products of a given phase fulfil the requirements established at the previous phase. [IEEE]

VTP –Validation Test Procedures

VTR –Validation Test Report.

walk through–A software inspection process conducted by the peers of the developer to evaluate a software item. Although usually associated with code examination, this process is also applicable to the software requirements and software design.

The major objectives of the walk through are to find defects (e.g., omissions, unwanted additions, and contradictions) in a specification or other product and to consider alternative functionality, performance objectives, or representations. [IEEE]

waterfall model–A software development lifecycle strategy that partitions the project into manageable phases:

(1) requirements, design, implementation, and test

(2) establishes milestones, documents, and reviews at the end of each phase. In the model, the successful completion of one lifecycle phase in the waterfall chart corresponds to the achievement of the counterpart goal in the sequence of software engineering goals for the software process. The waterfall model recognizes the iteration between phases. [IEEE]

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

REFERENCES FOR GLOSSARY

[CAN3-N286.2-86] CAN3-N286.2-86 "Design Quality Assurance for Nuclear Power Plants".

[CAN/CSA-Q396.1.1-89] CAN/CSA-Q396.1.1-89 "Quality Assurance Program for the Development of Software Used in Critical Applications".

[ESA] European Space Agency, PSS-05-0, "ESA Software Engineering Standards", Issue 2, February 1991.

[IEC 65A(Secretariat)94] Software for Computers in the Application of Industrial Safety Related Systems".

[IEEE] Dorfman, M., Thayer, R.H., "Standards, Guidelines, and Examples on System and Software Requirements Engineering", Glossary, IEEE Computer Society Press, 1990.

[MOD 00-55] MOD Interim Defence Standard 00-55 (Draft), "Requirements for the Procurement of Safety Critical Software in Defence Equipment".

[MOD 00-56] MOD Interim Defence Standard 00-56 (Draft), "Requirements for the Analysis of Safety Critical Software".

[MYERS] Myers, G.J., "The Art of Software Testing" New York, N.Y. Wiley-Interscience, July, 1979.

[TESTING] Ould, M.A., Unwin, C. (ed), "Testing in Software Development", British Computer Society Monograph, Cambridge University Press, 1988.

[PRESSMAN] Pressman, R.S., "Software Engineering", New York, McGraw-Hill Book Company, 1987.

[DeMarco] De Marco, T., "Structured Analysis and System Specification", Prentice-Hall, Englewood Cliffs, NJ, 1979.

[Ward&Mellor] Ward, P.T, Mellor, S.J, "Structured Development for Real-Time Systems", Yourdon Press (A Prentice-Hall Company), Englewood Cliffs, NJ, 1985.

Page left blank intentionally

In 366655
Appr. 30/08

KEYWORD: Software

SUBJECT: Software Engineering of Category III Software

APPENDIX A
Requirements For The Computer System Requirements Document

This appendix defines the requirements to be met by the contents of the Computer System Requirements (CSR) documentation.

The CSR documents define the requirements that must be met by the computer system. There may be a single requirements document for the whole computer system, or the requirements may be contained in several documents.

The CSR ***shall***:

(a) Define the functional, performance, safety, reliability, and maintainability requirements of the computer system, clearly identifying the safety requirements.

(b) Define the scope and boundary of the computer system and define the interfaces to other systems. It ***shall*** contain or reference a description of the problem domain including natural restrictions on the required functions.

(c) Define human interface requirements.

(d) Define all accuracy requirements and tolerances.

(e) Define any constraints placed on the design options.

(f) Define any Quality Assurance and Software Quality Assurance (SQA) requirements to be met by the computer system. Include or reference the categorization analysis that provides the rationale for choosing the SQA requirements.

(g) Define the design quality objectives, if different to those in Section E.1, and their relative priority.

(h) Define anticipated changes to the computer system based on an analysis of experience with changes to similar systems and on projections of future needs.

(i) Provide a clear definition of terms.

(j) Explicitly identify a comparable reference design or the absence of any.

(k) Contain no requirements that are in conflict with each other.

(l) Define each requirement uniquely and completely in one location to prevent inconsistent updates and to facilitate easy referencing by subsequent documents and verification processes.

(m) Contain or reference a revision history.

Page left blank intentionally.

dkt 366653
Appr. 3a/08

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

APPENDIX B
Requirements For The Development Process Outputs

This appendix defines the requirements which the outputs from the development processes must meet to be of acceptable quality. The outputs are:

(a) Computer System Design

(b) Computer Hardware Design

(c) Predeveloped Software Description

(d) Software Requirements Specification

(e) Software Design Description

(f) Source Code

The following detailed requirements reference, using brackets "[...]", the quality attribute(s) from Appendix E which they are intended to address.

B.1 COMPUTER SYSTEM DESIGN (CSD)

The CSD documentation defines the functional decomposition, the required predeveloped software, and the hardware architecture of the computer system. There may be a single computer system design document for the whole computer system, or the information may be divided into several documents.

The requirements provided below address only computer system design issues related to software engineering. *Specifically, requirements related to hardware engineering are* ***not*** *addressed.*

B.1.1 Functional Decomposition

The CSD ***shall***:

(a) Accurately define the computer system context. It ***shall*** define all inputs to the computer system ("monitored variables") and outputs from the computer system ("controlled variables"). A context diagram ***shall*** be provided. [Understandability, Verifiability]

(b) Decompose the computer system into functional subsystems with logical inputs and outputs, and uniquely identify the functional subsystems. [Understandability, Correctness, Structuredness, Modularity]

(c) Define the logical interfaces between the functional subsystems. Diagrams illustrating the interfaces between functional subsystems ***shall*** be provided. [Understandability, Verifiability, Correctness]

(d) Allocate system requirements to the functional subsystems. [Completeness, Verifiability, Modularity, Consistency, Understandability]

(e) Define the reliability requirements for each functional subsystem. [Robustness, Predictability, Correctness]

(f) Define the accuracy requirements and tolerances. [Correctness, Predictability]

(g) Define any constraints placed on the software design options. [Consistency]

(h) For each functional subsystem, define which functions must be easy to change or reconfigure. [Modifiability]

(i) Map the functional subsystems to one or more computers or non-programmed hardware, predeveloped software and custom developed software. [Understandability, Verifiability, Completeness]

(j) Identify and describe significant failure modes of the system. Provide an overview of required fault-tolerance and graceful degradation features provided by the system hardware architecture.

APPENDIX B (Continued)
Requirements For The Development Process Outputs

Examples would include hot-standby computers, warm-restart functions, backups, etc. Describe or reference the failure, restart, and resynchronization procedures and requirements. [Robustness, Predictability]

B.1.2 Hardware Architecture

For function subsytems allocated to hardware, the CSD *shall*:

(a) Identify each computer, physical interface between computers, and non-programmed hardware devices within the computer system. Diagrams of the computer system hardware architecture ***shall*** be provided. [Understandability, Completeness, Verifiability]

(b) Describe the general high-level characteristics of each computer, including types and approximate numbers of inputs, outputs, and I/O devices, approximate amount of memory and mass storage, approximate CPU performance, and other relevant information. The definition of required computer capabilities ***shall*** be sufficient to allow software requirements to be specified with confidence that the computer equipment will be sufficient to support them. [Completeness, Correctness]

(c) Describe the general high-level characteristics of each physical interface between computers, including type of communications link, performance, determinism and other relevant information. [Completeness, Predictability]

(d) Describe the general high-level characteristics of each non-programmed hardware device within the computer system. Identify the relevant design documentation for the device. [Completeness, Consistency].

B.1.3 Predeveloped Software

For functional subsystems allocated for predeveloped software, the CSD ***shall*** identify the types of predeveloped software to be used in the target system. This includes operating systems, database managers, libraries, compilers, etc. [Understandability, Completeness, Verifiability]

B.2 COMPUTER HARDWARE DESIGN (CHD)

The CHD documentation describes the computer hardware. There may be a single computer hardware design document for the whole computer system, or the information may be divided into several documents. References to the manufacturer's technical specifications may be used.

The requirements provided below address only computer hardware design issues directly related to software engineering. Other information may be required to engineer the hardware. Each of the requirements addresses the correctness and completeness quality attributes.

The CHD ***shall***, for each computer in the computer system, describe the computer in detail, including items such as:

(a) Bus type and speed,

(b) Hardware-designated address mapping,

(c) Direct Memory Access (DMA) capabilities,

(d) Hardware interrupts and interrupt controller,

(e) CPU type, instruction set, and speed,

(f) Coprocessor types and speeds,

(g) Memory types, sizes, and speeds,

(h) Memory management facilities, including caching,

(i) Mass storage device types, sizes, and speeds,

(j) I/O devices, including controller types, specifications, addressing, protocols, etc.,

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

APPENDIX B (Continued)
Requirements For The Development Process Outputs

(k) Required initialization logic,

(l) Required diagnostic facilities,

(m) Power up, power fail, and restart logic,

(n) Other information as required to completely characterize the software interface to the computer hardware.

B.3 SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

The SRS contains all requirements from the CSD which are relevant to the software as well as all other software specific requirements which arise due to the environment. Normally there will be one SRS for each functional subsystem. Alternative organizations are possible as long as the SRS is arranged to facilitate mapping to the functional subsystems defined in the CSD.

B.3.1 Requirements

The SRS *shall*:

(a) Describe all requirements from the CSD which are relevant to the software. [Completeness]

(b) Describe all additional requirements. [Completeness]

(c) Describe any software implementation design constraints. This might include the processors on which the software is required to execute, and the predeveloped software required to be used. [Completeness]

(d) Describe the logical inputs to and the logical outputs from the functional subsystem. [Completeness]

(e) Describe the required behaviour of the software in terms of what the software must do. [Completeness, Correctness, Verifiability]

(f) Describe the timing tolerances and the accuracy requirements by specifying the allowable deviation from the specified behaviour of the output variables. [Completeness]

(g) Describe the characteristics of the logical and physical inputs and outputs addressing such issues as types, formats, units, and valid ranges. [Completeness]

(h) Specify the response to exception or error conditions. [Predictability]

(i) Identify all those requirements for which future changes are anticipated. [Modifiability]

(j) Describe requirements for fault tolerence and graceful degradation. [Robustness]

B.3.2 Constraints on the SRS

The SRS *shall*:

(a) Be written strictly in terms of requirements and design constraints; it ***shall*** limit the range of valid solutions, but ***shall*** not insist on a particular design. [Completeness]

(b) Be written according to SPH guidelines. [Correctness, Completeness, Consistency]

(c) Use standard terminology and definitions throughout the document. [Consistency]

B.4 SOFTWARE DESIGN DESCRIPTION (SDD)

The SDD document is a representation of the software design. Normally the SDD will be organized into an overview of the software architecture, sections or subdocuments corresponding to each of the functional subsystems. Alternative organizations are possible as long as the SDD is arranged to facilitate mapping to the SRS. The SDD will be the primary document issued to the client to allow understanding, use and maintenance of the software.

KEYWORD: Software

SUBJECT: Software Engineering for Category III Software

APPENDIX B (Continued)
Requirements For The Development Process Outputs

B.4.1 Software Architecture

The software architecture section of the SDD ***shall***:

(a) Describe a software architecture which meets the requirements of the SRS to a level of detail that requires no further refinement of the predeveloped software, tasks, databases, intertask communications, or input/output. Document the software architecture using appropriate diagrams. If more than one task is used, provide the following diagrams for each computer:

(1) a task activation hierarchy diagram, showing which tasks activate which other tasks (and which are scheduled by the operating system),

(2) diagrams showing all intertask communications, and the interface to the computer system input/output. [Understandability, Verifiability, Maintainability]

For the purposes of this requirement, each instance of each task ***shall*** be identified explicitly. Both developed and predeveloped software ***shall*** be included in this description.

(b) Describe how each predeveloped software component incorporated into the software relates to the developed software. Provide diagrams as necessary to describe the configuration of the software. Define the detailed interface between the predeveloped software and the developed software. References to the manufacturer's documentation may be used. [Understandability, Verifiability, Maintainability]

(c) Identify which components of the predeveloped software are used in the target application, which are included but not used, and which are not included. [Understandability, Maintainability]

(d) Provide a mapping between functional subsystems, tasks, predeveloped software and computers. Understandability, Verifiability, Maintainability]

(e) Describe each task. [Understandability, Verifiability, Maintainability]

(f) Describe each data structure. [Understandability, Verifiability, Maintainability]

(g) Describe each intertask communications service. [Understandability, Verifiability, Maintainability]

(h) Describe each input/output device. [Understandability, Verifiability, Maintainability]

(i) Describe each library to be shared between tasks. [Understandability, Verifiability, Maintainability]

(j) Describe how the executable software for each functional subsystem including predeveloped software and computer is organized in the target computer system, for example, directory structures or memory maps. [Understandability, Verifiability]

(k) Identify the language(s) and translator(s) to be used. [Verifiability]

B.4.2 Software Detailed Design

This section of the SDD ***shall***:

(a) For each task or library identified in the software architecture section, describe a software design which meets the relevant requirements of the SRS to a level of detail that requires no further refinement in the code of the module structure, module functions, module interfaces, libraries and data structures.

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

APPENDIX B (Continued)
Requirements For The Development Process Outputs

Provide the decomposition of the task or library into design entities. Provide diagrams showing the program calling hierarchies. Where object-oriented programming techniques are used, provide diagrams of the class inheritance hierarchies. [Understandability, Verifiability, Maintainability]

(b) Provide a "module guide" containing the following information for each module: [Understandability, Verifiability, Maintainability]

(1) unique identifying name,

(2) the purpose of the module,

(3) the data structure(s) maintained by the module,

(4) the list of programs contained in the module,

(5) any required resources (elements external to the design, e.g., predeveloped software, devices, software services (libraries), operating system services, processing resource requirements),

(6) the programming language(s) to be used.

(c) Provide the following information for each program: [Understandability, Verifiability, Maintainability]

(1) unique identifying name,

(2) program type (e.g., main program, subroutine, function, macro, interrupt handler),

(3) the purpose of the program (what requirement it satisfies),

(4) program interface (calling sequence and passed parameters).

(d) Provide information for each data structure and data element (possibly in the form of a data dictionary):

(1) unique identifying name,

(2) data type (e.g., integer, Boolean, floating point, structure, file),

(3) the purpose of the data structure (what data it contains),

(4) the range of validity.

(e) Describe how the software for each task or library is built or generated. [Understandability, Verifiability]

B.4.3 Other Requirements To Be Met By The SDD

The SDD ***shall***:

(a) Describe error handling which is not specified in the SRS. [Robustness]

(b) Describe how relevant experience from previous systems defined in the CSD has been factored into the design. [Correctness]

(c) Identify all functions contained in the SDD which are outside the scope of the SRS.

(d) Identify each function in the design so that it can be uniquely referenced by the code. The identifiers ***shall*** be consistent with those used in the code.

(e) Contain or reference a revision history which identifies all modifications to the design and the rationale for these changes.

(f) Use standard terminology and definitions throughout the document. [Consistency]

B.5 SOURCE CODE

The source code is a complete and accurate translation of the design described in the SDD. It includes both executable statements and data declarations.The source code ***shall*** precisely implement the design as described in the SDD. [Verifiability, Understandability, Correctness].

Page left blank intentionally.

TN 366653
Appx 3 of 8

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

APPENDIX C
Requirements For The Verification Process Outputs

This appendix defines the requirements which the outputs from the verification processes must meet to be of acceptable quality. The outputs are:

(a) Computer System Design Review Report,

(b) Software Requirements Review Report,

(c) Software Design Review Report,

(d) Subsystem Test Procedures,

(e) Subsystem Test Report,

(f) System Integration Test Procedures,

(g) System Integration Test Report,

(h) Validation Test Procedures,

(i) Validation Test Report.

Many of the requirements on the review and verification reports are generic and are listed in Subsection C.1.

Those requirements which are specific to the various review and verification reports are dealt with in separate subsections.

The requirements on the test procedure outputs are presented individually in the following subsections.

The requirements for all test reports are presented in Subsection C.8 as a common set of requirements.

C.1 GENERIC VERIFICATION REPORT REQUIREMENTS

The Verification Reports have a common set of requirements. To be of acceptable quality, a report *shall*:

(a) Identify the versions of the relevant documents.

(b) Summarize the review or verification activities performed and the methods and tools used.

(c) Summarize the discrepancies.

(d) Summarize the positive findings.

(e) Describe the conclusions and recommendations.

(f) Identify the review or verification participants.

(g) Identify and comply with the applicable SPH requirements for Verification Reports.

(h) Identify or reference corrective action lists resulting from the review or test.

C.2 COMPUTER SYSTEM DESIGN REVIEW REPORT (CSDRR)

To be of acceptable quality, the CSDRR *shall*:

(a) Provide evidence that a review of the design has been performed.

(b) Provide evidence that the review has covered all requirements and design constraints in the CSR and all functional subsystems, computers, and interfaces in the CSD.

(c) Provide evidence that the review has covered all standards and procedures in the SPH applicable to the CSD.

C.3 SOFTWARE REQUIREMENTS REVIEW REPORT (RRR)

To be of acceptable quality, the RRR *shall*:

(a) Provide evidence that a review of the requirements has been performed.

KEYWORD: Software

SUBJECT: Software Engineering for Category III Software

APPENDIX C (Continued)
Requirements For The Verification Process Outputs

(b) Provide evidence that the review has covered all requirements and design constraints in the CSD and SRS.

(c) Provide evidence that the review has covered all requirements and design constraints appearing in the SRS which are not derived from the CSD.

(d) Provide evidence that the review has covered all standards and procedures in the SPH applicable to the SRS.

C.4 SOFTWARE DESIGN REVIEW REPORT (DRR)

To be of acceptable quality, the DRR *shall*:

(a) Provide evidence that a review of the design has been performed.

(b) Provide evidence that the review has covered all requirements and design constraints in the SRS and all programs, data structures, and data bases in the SDD.

(c) Provide evidence that the review has covered all standards and procedures in the SPH applicable to the SDD.

C.5 SUBSYSTEM TEST PROCEDURES (STP)

The Subsystem Test Procedures *shall*:

(a) Require all tests to be done with predeveloped software and test hardware that accurately represents the capabilities of the target, that are required for the functional subsystem.

(b) Define test cases to test each functional requirement in the SRS.

(c) Define tests to test the performance requirements as described in the SRS.

(d) Identify all resources used by the functional subsystem and define test cases which test the functional subsystem under conditions that attempt to overload these resources in order to determine if the functional and performance requirements defined in the SRS are met.

(e) Define tests to exercise any interfaces between the software and the hardware, and the software and the predeveloped software.

(f) Define tests to show that the functional subsystem meets its requirements under each hardware configuration and operational option.

(g) Define tests to test the ability of the functional subsystem to respond as specified in the SRS to software, hardware, and data errors.

(h) Define test cases which attempt to subvert any existing safety or security mechanisms.

(i) Describe expected results of each test case so that a pass/fail determination can be made as to the outcome of the test. The expected results *shall* be based on the information contained in the SRS.

(j) Identify all equipment (and its required calibration), tools, and support software required to perform the test and provide adequate setup and test execution instructions so that the test can be repeated by personnel who did not perform the original test. Each test procedure *shall* describe or reference how the executable code to be tested is built.

(k) Identify the SRS name and version.

(l) Provide a cross-reference between the sections of the SRS and the test procedures to document the test coverage.

(m) Comply with the applicable sub-tier standards and procedures in the SPH.

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

APPENDIX C (Continued)
Requirements For The Verification Process Outputs

C.6 SYSTEM INTEGRATION TEST PROCEDURES (SITP)

The System Integration Test Procedures ***shall***:

(a) Require all tests to be done with predeveloped software and hardware that is closely representative of the capabilities of the complete target system architecture described in the CSD.

(b) Define hardware and software components to be integrated and tested.

(c) Define the system integration testing strategy and plan to be used, e.g., Bottom-Up Integration, Top-Down Integration, Incremental Builds, etc.

(d) Define test cases to test that the interfaces between functional subsystems behave as specified in the CSD.

(e) Define test cases to test that the physical interfaces between computers behave as specified in the CSD.

(f) Define test cases to test that the functional subsystems perform in an integrated environment as specified in the CSD.

(g) Define tests cases to test that the system architectural features perform as specified in the CSD.

(h) Describe expected results of each test case so that a pass/fail determination can be made as to the outcome of each test from information provided in the CSD.

(i) Identify all equipment (and its required calibration), tools, and support software required to perform the test and provide adequate set-up and test execution instructions so that the test can be repeated by personnel who did not perform the original test. Each test procedure ***shall*** describe or reference how the executable code to be tested is built.

(j) Identify the CSD name and revision.

(k) Provide a cross-reference between the sections of the CSD components and the test procedures to document the test coverage.

(l) Comply with the applicable sub-tier standards and procedures in the SPH.

C.7 VALIDATION TEST PROCEDURES (VTP)

The Validation Test Procedures ***shall***:

(a) Require all tests to be done with predeveloped software and hardware that duplicates to the greatest practical extent the complete target system architecture.

(b) Define test cases to test each functional requirement in the CSR that is applicable to the system.

(c) Define test cases to test each performance requirement in the CSR that is applicable to the system.

(d) Define test cases, using dynamic simulation of input signals, to cover normal operation, anticipated operational occurrences, abnormal incidents, and accident conditions requiring system action as indicated in the CSR.

(e) Describe expected results of each test case so that a pass/fail determination can be made as to the outcome of the test from information provided in the CSR.

(f) Identify all equipment (and its required calibration), tools, and support software required to perform the test and provide adequate setup and test execution instructions so that the test can be repeated by personnel who did not perform the original test. Each test procedures ***shall*** describe or reference how the executable code to be tested is built.

APPENDIX C (Continued)
Requirements For The Verification Process Outputs

(g) Identify the CSR components and versions.

(h) Provide a cross-reference between the sections of the CSR components and the test procedures to document the test coverage.

(i) Comply with the applicable sub-tier standards and procedures in the SPH.

C.8 TEST REPORTS

A test report must document the results of each test activity.

The test report output from each test activity ***shall***:

(a) Identify or describe the tests performed.

(b) Identify the test procedures.

(c) Include the comparison of actual and expected test results as defined in the referenced test procedures.

(d) Summarize the discrepancies.

(e) Summarize the positive findings.

(f) Describe the conclusions and recommendations.

(g) Identify the date and time of the performance of each test.

(h) Identify the program, functional subsystem, or system, and the versions being tested.

(i) Identify the testers involved.

(j) Reference the detailed test results.

(k) Identify compliance with the applicable sub-tier standards and procedures in the SPH.

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

APPENDIX D
Requirements For The Support Process Outputs

This appendix contains the requirements which the outputs from the support process ***shall*** meet to be of acceptable quality. The outputs are:

(a) Software Development Plan (SDP)

(b) Standards and Procedures Handbook (SPH)

D.1 SOFTWARE DEVELOPMENT PLAN (SDP)

The SDP provides the comprehensive plan for the management of the software engineering process.

The requirements of the SDP are subdivided into requirements for the portions which make up the SDP:

(a) Project Plan Portion

(b) Development Plan Portion

(c) Verification Plan Portion

(d) Documentation Plan Portion

(e) Configuration Management Plan Portion

The SDP ***shall not*** duplicate information contained in overall project management documentation, but rather ***shall*** reference it. The SDP ***shall*** be revised when there are major changes to either the software scope of work or to the project organizational structure.

In the following requirements, the term "provide" allows developers to use either descriptions or references to supply the required information.

D.1.1 Project Plan Portion of SDP

The project plan portion of the SDP describes the scope of the software engineering effort, the organization and responsibilities, and the key milestones and dates based on external detailed budget and schedule documentation used for monitoring the work effort.

The project plan portion of the SDP ***shall***:

(a) Identify the CSR

(b) Identify any overall project management plans, and describe the relationship of the SDP to them.

(c) Adopt a specific software life cycle model, such as waterfall, iterative waterfall, evolutionary builds, pre-planned product improvement, etc., as a focus for planning that a systematic software engineering process is followed over the entire life of the software. If iterations are planned, then the model ***shall*** show the intended number of iterations. In addition, it ***shall*** treat development and maintenance as an integral, continuous, and interactive process.

(d) Describe the organizational structure for the project and relationships to external organizational units (including the user), including the scope, authority and responsibility of each. Show all relevant job positions and the number of resources required within the project and list the independence requirements of each job position.

(1) Define the following independence roles:

- Developer
- Verifier
- Validator

(2) Responsibility for the development, verification, and support processes ***shall*** be assigned to the defined roles as described in Table D-1.

(3) Verifiers have primary responsibility for directing verification processes but developers may be involved in the performance.

Verification ***shall not*** be performed by the same person who developed the item being verified.

KEYWORD: Software

SUBJECT: Software Engineering for Category III Software

APPENDIX D (Continued)
Requirements For The Support Outputs

TABLE D-1

PROCESS	ROLE
Computer System Design	Developer
Software Requirements Definition	Developer
Software Design	Developer
Code Implementation	Developer
Computer System Design Review	Verifier
Requirements Review	Verifier
Design Review	Verifier
Sub-System Testing	Verifier
System Integration Testing	Verifier
Validation Testing	Validator
Planning	Any
Configuration Management	Any

(4) Developers ***shall not*** be involved in roles assigned to Validators.

(e) Define the overall software development effort and cost. Document this estimate and rationale used to determine the estimate for future use and review.

(f) Provide the project schedule. Identify key milestones and hold points beyond which no further activities can be started until the necessary completion criteria are successfully met for the prerequisite activities.

(g) Include a policy statement that the SDP ***shall*** be followed by all personnel who are responsible for the production of an output required by the SDP.

(h) Include a policy statement that the personnel who produce each output required by the SDP have primary responsibility for the output's quality.

(i) Mandate the project-specific standards and procedures in the SPH based on the requirements in Appendix D.2.

(j) Identify training requirements for the project.

D.1.2 Development Plan Portion of SDP

The development plan portion of the SDP describes the approach to development, key design and implementation issues, and identifies the support tools and facilities.

The development plan portion of the SDP ***shall***:

(a) Partition the development effort into uniquely identifiable development processes with well defined inputs and outputs (see Table D-1).

(b) Specify the approach and methodologies to be used for the development processes.

(c) Specify facilities, tools, and aids to be used during software development.

(d) Identify any key design and implementation issues and preliminary studies, simulation modelling, and/or prototyping required to resolve them.

D.1.3 Verification Plan Portion of SDP

The verification plan portion of the SDP describes the approach to the verification of the software product. It identifies the verification activities and support tools and facilities required to perform the verification.

The verification plan portion of the SDP ***shall***:

(a) Partition the verification effort into uniquely identifiable verification processes with well defined inputs and outputs (see Table D-1).

(b) Specify the approach and methodologies to be used for the verification processes.

(c) Specify facilities, tools, and aids to be used during verification.

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

APPENDIX D (Continued)
Requirements For The Support Process Outputs

(d) Require that verification activities ***shall not*** start until the outputs being verified are completed and placed under configuration control.

(e) Identify the qualification requirements for predeveloped software for the target system and software tools.

D.1.4 Documentation Plan Portion of SDP

The documentation plan portion of the SDP lists and provides summary descriptions of all project documentation related to software. It also identifies responsibility for producing, reviewing, approving, and issuing documents. At the very least, the plan covers all document outputs identified in Table D-1.

The documentation plan portion of the SDP ***shall***:

(a) List and briefly describe all project documents and records that are a result of the software project covered by the SDP. It ***shall*** distinguish between deliverable end-user documentation and non-deliverable documentation.

(b) Provide a list of all deliverable software, test software, and support software.

(c) Provide a list of all deliverable documents obtained from external suppliers as they become known.

(d) Identify responsibility for producing, reviewing, approving, filing, and issuing documents.

(e) Identify facilities, tools and aids used to produce, manage, and publish documentation.

D.1.5 Configuration Management Plan Portion of SDP

The configuration management plan portion of the SDP identifies the configuration management activities, the organizational structure and independence requirements, and the support tools and facilities.

The configuration management plan portion of the SDP ***shall***:

(a) Uniquely identify all categories of configuration items that form a developmental baseline for software configuration management: application software (source, binary, library, document), software media, all deliverable and non-deliverable documents, support tools (compilers, linkers, operating systems), data, test software, command files or scripts, and predeveloped software. Specify the identification scheme within each type.

(b) Specify when and under what conditions all configuration items come under change control. As a minimum, every document and piece of source code ***shall*** go under change control before it is given to an independent party for verification purposes.

(c) Require that each change be implemented starting at the highest affected output and proceeding to subsequent outputs (e.g., CSR to CSDDID to SRS to SDD to source code) and that each undergo the same degree of verification as provided for the original outputs.

(d) Define a mechanism to periodically review, classify, and adjudicate submitted change requests. The mechanism must allow for participation by members representing each organizational unit that is responsible for one of the project outputs identified in Table D-1.

(e) Identify a centralized support library to provide storage for, and controlled access to, software and documentation in both human-readable and machine-readable forms as well as records to support audit and review of the software configuration.

(f) Identify a mechanism for analysis of the errors detected during verificatio as an aid to on-going improvement of standards and procedures in the SPH and the various test procedures (STP, SITP, VTP).

KEYWORD: Software

SUBJECT: Software Engineering for Category III Software

APPENDIX D (Continued)
Requirements For The Support Outputs

(g) Identify how to integrate predeveloped software and documentation into the adopted configuration management system.

D.2 STANDARDS AND PROCEDURES HANDBOOK (SPH)

The purpose of the SPH is to document the sub-tier standards, procedures, work practises, and conventions adopted by a specific project to meet the requirements of this standard. The requirements of the Standards and Procedures Handbook (SPH) are subdivided into requirements for the portions which make up the SPH:

(a) Project Plan Standards and Procedures

(b) Development Standards and Procedures

(c) Verification Standards and Procedures

(d) Documentation Standards and Procedures

(e) Configuration Management Standards and Procedures

All standards and procedures ***shall*** be consistent with requirements specified in this standard.

D.2.1 Project Plan Standards and Procedures

The project plan standards and procedures ***shall*** identify standards for the SPH.

D.2.2 Development Standards and Procedures

The development standards and procedures ***shall***:

(a) Provide standards, procedures, and guidelines for each of the development processes and outputs listed in Table 1 (except computer hardware design and software acquisition, which are outside the scope of this standard). The processes include:

(1) Computer System Design

(2) Software Requirements Definition

(3) Software Design

(4) Code Implementation

(b) For each development process listed above and the corresponding output, provide the following sub-tier standards or guidelines:

(1) Guidelines describing the methodologies, techniques, and principles to be employed during the development process.

(2) Guidelines identifying the tools to be used during the development process, and any procedures that must be followed in using them.

(3) Standards describing the organization and contents of the development output.

(c) Provide the following guidelines for the design process:

(1) Provide guidelines on structuring the software, including Structured Design, Information Hiding, Structured Programming, complexity metrics, and heuristics to be followed. [Modularity, Structuredness, Modifiability, Understandability].

(2) Provide guidelines on plausibility checking and the handling of error conditions. [Robustness]

(3) Describe under which circumstances functions not specified in the SRS may be performed by the design, and what documentation requirements must be met. [Correctness]

(d) Define coding guidelines for:

(1) Code layout and formatting,

(2) Commenting,

(3) Naming conventions,

(4) Listings, maps, and cross references,

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

APPENDIX D (Continued)
Requirements For The Support Process Outputs

(5) Documentation of revisions,

(6) Structured programming and other aspects of control flow,

(7) Algorithms and expressions,

(8) Data structures and types, variables, and constants,

(9) Run-time error checking,

(10) Language-dependent guidelines,

(11) Machine-dependent guidelines.

D.2.3 Verification Standards and Procedures

The verification standards and procedures *shall*:

(a) Provide standards, procedures, and guidelines for each of the verification processes and outputs listed in Table D-1. The processes include:

(1) Computer System Design Review,

(2) Software Requirements Review,

(3) Software Design Review,

(4) Subsystem Testing,

(5) System Integration Testing,

(6) Validation Testing,

(b) For each verification process identified above and the corresponding outputs, provide the following standards or guidelines:

(1) Guidelines describing the methodologies, techniques, and principles to be employed during the verification process.

(2) Guidelines identifying the tools to be used during the verification process, and any procedures that must be followed in using them.

(3) Standards describing the organization and contents of the verification output or outputs.

D.2.4 Documentation Standards and Procedures

The documentation standards and procedures *shall*:

(a) Identify a standard document style and format to be used for all documents. The style and format guidelines *shall* be chosen to make the documents consistent, understandable, reviewable, and maintainable.

(b) Define a comprehensive glossary to ensure common understanding and consistent usage of project-specific and industry accepted terminology. The use of unique terminology and definitions *shall* be minimized.

D.2.5 Configuration Management Standards and Procedures

The configuration management standards and procedures *shall*:

(a) Specify guidelines to handle and document exception cases or problems arising during software development and verification.

(b) Identify procedures for change requests to fix problems or exception cases.

(c) Identify procedures for unique identification of:

(1) software changes,

(2) software and software components,

(3) release, version, and update designations,

(4) documentation,

(5) media.

KEYWORD: Software

SUBJECT: Software Engineering for Category III Software

APPENDIX D (Continued)
Requirements For The Support Outputs

(d) Define procedures for issuing, analyzing, estimating the impact of, classifying, approving or disapproving, distributing, scheduling implementation of and tracking software change requests to completion.

(e) Identify procedures for support library operation (i.e., security, disaster recovery, archive maintenance, protection, access control, retention period, change history, withdrawal, submission, issue, and approval).

(f) Define procedures for change request documentation and configuration management status reports in order to identify the most current configuration.

(g) Define procedures for approving and issuing software releases to the use.

(h) Define procedures for verification activities (including regression testing) following a change.

(i) Identify tools to be used for configuration management and procedures for their use.

KEYWORD: Software

SUBJECT: Software Engineering For Category III Software

APPENDIX E
Rationale For The Requirements Of This Standard

This standard provides measurable requirements to evaluate the quality of a software product, which consists of all outputs listed in Table 1. These measurable requirements are based on concepts of quality attributes and software engineering process principles.

A software product is considered acceptable if it is shown to satisfy a set of quality objectives defined in the Computer System Requirements as per Section E.1. The quality objectives can be refined in terms of quality attributes specified in Section E.2 which provide more concrete requirements for software quality.

The quality attributes are built into the product by adhering to the software engineering process defined by this standard. This process is governed by a set of fundamental software engineering principles specified in Section E.3. This provides assurance that the quality objectives are met.

E.1 QUALITY OBJECTIVES

The quality objectives are general statements about the relative importance of the various quality attributes. For instance, for a given project, portability between computer hardware platforms may be an objective, and this may be more important than efficiency.

The basic quality objectives are as follows:

Functionality: Documented assurance ***shall*** be provided that functions have been defined fully, implemented correctly, and verified completely. This objective implies the quality attributes Completeness, Correctness, Understandability, and Verifiability.

Reliability: The software ***shall*** perform its required functions such that the probability of it successfully performing those functions is consistent with the reliability requirements of the system of which it is a part. This objective implies the quality attributes Predictability and Robustness.

Maintainability: The software ***shall*** be structured so that those items most likely to require modification can be changed efficiently and reliably.

This implies the quality attributes Modifiability, Understandability, Verifiability, Modularity, Structuredness, Consistency.

Reviewability: The software ***shall*** be written and documented so that it can be understood, used and maintained by the user. This leads to all of the same quality attributes as "maintainability" except "modifiability."

Additional quality objectives to be considered are:

Portability: The software ***shall*** be capable of being transferred to different environments to the degree required by the users.

Usability: The software ***shall*** be easy to use so that effort by human users to obtain the required service is minimal. [Understandability, Predictability]

Efficiency: The software ***shall*** achieve an acceptable level of performance within the constraints of hardware resources (e.g., memory, CPU speed).

The CSR defines quality objectives for a specific project, and provides guidance about priorities and tradeoffs. This facilitates tailoring individual project quality requirements in the SPH guidelines.

E.2 QUALITY ATTRIBUTES

The requirements for the output of each software engineering process are provided in the appendices. The requirements for each development output are tagged with the quality attribute(s) which they address. These quality attributes are listed in Table E-1 and defined in the glossary.

The verification and support processes in this standard provide assurance that the quality objectives are satisfied.

KEYWORD: Software

SUBJECT: Software Engineering for Category III Software

APPENDIX E (Continued)
Rationale For The Requirements Of This Standard

TABLE E-1
Quality Attributes Used in this Standard

- Completeness
- Correctness
- Understandability
- Verifiability
- Robustness
- Predictability
- Modifiability
- Modularity
- Structuredness
- Consistency

E.3 SOFTWARE ENGINEERING PRINCIPLES

To achieve the quality objectives listed in, the software engineering process follows a set of principles. These principles, together with the quality attributes they address, include:

(a) The overall structure of the computer system and the software architecture ***shall*** be documented using rigorous techniques, including diagrams. [Understandability, Verifiability, Completeness, Correctness, Structuredness]

(b) The structure of the software ***shall*** be based on the use of recognized software engineering practices (e.g., Information Hiding, Structured Analysis and Design, Object-oriented Design, Top-Down Decomposition, Structured Programming, etc.) [Understandability, Modifiability, Structuredness, Modularity]

(c) The outputs from each development process ***shall*** be reviewed to verify that they comply with the requirements specified in the inputs to that process. [Correctness]

(d) Verification of the software ***shall*** be carried out throughout its entire life. Any changes ***shall*** be verified to the same degree of rigour as the original development. [Correctness]

(e) Engineering of the software ***shall*** follow a planned and systematic process over the entire life of the software. [Correctness, Modifiability, Verifiability]

(f) Configuration management ***shall*** be maintained throughout the entire life of the software to ensure up-to-date and consistent software and documentation. [Consistency, Correctness, Verifiability]

(g) The software engineering process ***shall*** be capable of being tailored for different sets of quality objectives, different projects varying in size and complexity and the procurement of predeveloped software developed to commercial standards.

These principles are intended to cost-effectively provide a level of assurance that the delivered software will meet the requirements and quality objectives described in the CSR.