INFO-0669

REPORT

Software Quality Assurance: Practical Procedures to Meet CSA Standards

by

M.B. Carver

Prepared for
the Atomic Energy Control Board
under contract 96-7,
Software Quality Assurance Program
Ottawa, Canada

April 1997

AECB Catalogue number INFO-0669

SOFTWARE QUALITY ASSURANCE: PRACTICAL PROCEDURES TO MEET CSA STANDARDS

A report prepared by M.B. Carver. Work completed under Atomic Energy Control Board contract 96-7, Software Quality Assurance Procedures. Contract Officer Dr. M. El-Hawary.

ABSTRACT

Standards for software quality assurance (SQA) are a requisite for nuclear safety analysis, and organisations involved in these analyses are required to document their SQA procedures. The intent of this report is to provide a stand-alone document that, following the intent of the CSA/SQA standards and using the same definitions of those documents, provides pragmatic guidelines for writing a set of procedures that would meet AECB requirements for the control and application of computer programs used in safety analysis. Since some procedures are already in place in the industry, the document can be used to quickly assess to what degree an existing procedure conforms to the requirements.

RÉSUMÉ

Les normes d'assurance de la qualité des logiciels sont essentielles pour l'analyse de sûreté nucléaire, et les organismes qui participent à ces analyses sont tenus de documenter leurs procédures d'assurance de la qualité des logiciels. Le but du présent rapport est de fournir un document unique qui, suivant l'objectif des normes d'assurance de la qualité des logiciels de la CSA et utilisant les mêmes définitions que celles de ces documents, donnerait des lignes directrices pratiques pour la rédaction d'un ensemble de procédures répondant aux exigences de la CCEA pour le contrôle et l'application de programmes informatiques utilisés pour l'analyse de sûreté. Comme certaines procédures sont déjà en place au sein de l'industrie, le document peut servir à évaluer rapidement dans quelle mesure des procédures existantes respectent les exigences.

TABLE OF CONTENTS

1. Background

1.1 AECB Requirements for Software Quality Assurance

The AECB Consultative Document C-6.R1 [1], entitled Requirements for the Safety Analysis of CANDU Nuclear Power Plants, states that a quality assurance program shall be established in accordance with CAN/CSA-N286.0-92 and CAN/CSA-286.2-86, modified for application to safety analysis.

With respect to reporting requirements for safety analyses, the document calls for an auditable quality assurance program, and sufficient documentation to permit verification of all analyses by an independent team of qualified analysts. With respect to computer codes utilised, the document further states the following :

"All computer codes shall be documented in accordance with codes and standards that are agreed to by the AECB."

"This documentation should include a functional specification, quality assurance plan, a configuration management plan, a version description, a user's manual, training material, a software design description, and validation reports, computer listings and data.".

This document further relates these requirements to the CSA standards.

1.2 The CSA Standards

While there are a number of standards on general quality assurance (QA) and on software quality assurance (SQA) in particular, the CSA standards cover most necessary attributes of an effective quality assurance program. In particular the Canadian Standard Association series of N286 documents define quality assurance standards for activities related to a nuclear power plant project.

CAN/CSA N286.0-92 Covers Overall Quality Assurance
CAN/CSA N286.2-86 Covers Design Quality Assurance
CAN/CSA N286.1,2,3,4,5 Cover respectively Procurement, Construction, Commissioning, and Operation
CSA N286.7-94 Covers Software Quality Assurance

CSA N286.7-94 has been issued as a preliminary standard entitled: Quality Assurance of Analytical, Scientific and Design Computer Programs for Nuclear Power Plants. The standard addresses computer programs for analytical, scientific and design applications for safety-related systems and components, and applies to programs originating in the nuclear industry and programs procured from external vendors. This standard has been under development for several years and is still under going revision.

A draft revision dated March 1996 has also been consulted in preparing this document, this revision is referred to as N286.7-96. Normally, a draft revision would not be considered until it is issued, however N286.7-96 is considered important for two major changes:

i) the 96 revision has substantially condensed the requirements and eliminated repetition and inconsistencies between the sections on procedural matters and the section on documentation.

ii) this revision includes guidelines on the control of computer programs that were developed prior to 1996, but are undergoing further development. (N286.7-94 merely states that such programs should be treated as special cases.)

The first change is useful to the goal of writing practical procedures. The second is essential, as most of the codes used in safety analysis fall in this category, and it is necessary to plan and control them in a manner similar to that required for design and development of new software. While the provision of these guidelines is a useful step, this version of the standard considers all pre-existing software to be exempt from the design and development documentation that is required for new software. This appears not to impose any quality criteria on existing programming, which is somewhat unsatisfactory.

The AECB finds that the 96 version of the N286 standard does not yet fully satisfy their requirements for safety analysis, but have requested that this document address this standard as the most practical currently available for safety analysis.

The CAN/CSA Q396 series (1989) has four documents that address SQA programs for i) the development of Software and ii) the use of Previously developed software in a) Critical Applications and b) Non-Critical Applications. These standards cover many of the topics included in N286.7, but concentrate more on the SQA aspects of the customer client relationships.

This document refers to CSA N286.7 and the CSA Q396 series collectively as the CSA SQA standards.

1.3 Software Quality Assurance Procedures

A requirement under the CSA series, in fact a requirement essential to all software quality assurance standards, is that each organization must establish the procedures it follows for software quality assurance to control work with computer programs in a manner consistent with the standard. The development and documentation of SQA procedures is the necessary first step in the SQA trail, equally important is the implementation of these procedures to ensure that all computing work not merely follows the procedures, but can be clearly demonstrated to do so.

While the CSA SQA standards have been available in some form for several years, the implementation of their recommendations within the industry is still in its formative stages. It is fair to say that most computer codes now have some procedures in place governing their development and use. Certainly some of the major codes are now developed and applied under clearly established software quality assurance procedures, however many others have only limited procedures defined. Since N286.7 has been evolving, the SQA procedures that do exist do not necessarily follow all its recommendations in a satisfactory manner, hence this document can be used to review existing procedures and improve them where necessary

1.4 Objectives of Software Quality Assurance Procedures

The objectives of such procedures are to ensure that the safety analyses performed are appropriate, and are conducted by qualified personnel who use quality assured computer programs in an appropriate manner, and produce results that are credible, and can be reproduced and verified.

1.5 Purpose and Scope of This Document

The standards call for SQA procedures, but are not specific about their content or format. The purpose of this document is to provide pragmatic guidelines for writing a set of procedures that satisfy AECB requirements for the control and application of computer programs used in safety analysis, following the structure and intent of the CSA/SQA standards and using the same definitions of those documents. The document will be as brief as practicable in that it will not reproduce the text of the standards unless necessary for clarification.
It is hoped the document will also ease the process of creation, review and implementation of these procedures by defining a pragmatic approach suitable for implementation in a manner compatible with the everyday working environment in the industry. To this end the document contains a recommended format for an SQA procedures document, a discussion of each item in the contents, and basic templates for forms that address the requirements of the standards.

Since some procedures are already in place in the industry, the document also provides a format that can be used to quickly assess to what degree an existing procedure conforms to the requirements.

The document addresses specific items required by the standards and also includes recommendations for practice where the standards are not specific. Occasionally a recommendation is included for a good practice in an area not specifically addressed by the standards.

1.6 Structure of this Document

Section 2 of this document gives a recommended format for writing SQA procedures, with some discussion of each topic. All the CSA SQA standards were consulted while preparing the document. However as each standard has a somewhat different approach, the decision was made to align the topic sequence with that of N286.7, and reference the other standards. Appendix A contains a summary of the relationship between the topic sequence and all the CSA SQA standards.

In section 2, the topic sequence corresponds closely to N286.7, but some topics have been combined differently in an attempt to clarify their relationship.

Section 3 of this report contains a sample procedures document that was prepared by following the recommendations of Section 3.

The Appendices contain background information and a number of example forms that could be used to address some of the requirements of the standard. The intent in providing the forms is not to prescribe format, but merely to illustrate that the necessary SQA records can be documented in a relatively concise manner.

1.7 Definitions of Terms used in this Document

Appendix B lists definitions of terms used in this document. It contains all the definitions from N286.7 . *Some additional definitions are included to clarify the current document, these appear in italic font.*

1.8 SQA Procedures and Company Organisation

It is recognised that in large organisations, it may not be practical to get every unit of the organisation to follow exactly the same detailed procedures for software quality assurance, due to differences in mandate and current status of SQA amongst the units. It is expected that the organisation as a whole, or major divisions of the organisation will have a Quality Assurance Manual, whereas the functional units will develop and document procedures for software quality assurance.

This document lays out the necessary components required to establish procedures for the control and use of software in a particular functional unit. The definition of such a unit and its mandate is the responsibility of the organisation itself, but for the purposes of this document, the unit will be regarded as a working group that has the mandate to control the development and/or management and/or application of one or more computer codes.

The development, management and application of computer codes are in fact three distinct disciplines, and normally should be undertaken by different staff groups, although these groups may have common management.

2. Recommended Format and Contents for a Unit SQA Procedures Document

This section outlines the recommended format and content of a Unit SQA Procedures Document, i.e. a document describing SQA Procedures for a functional unit. Discussion is provided for each topic. For brevity, the Unit SQA Procedures Document will be referred to as USQAPD. Section 3 has a sample USQAPD that follows the same format.

In this section, items that appear in regular typescript are intended to represent the standards, either as reiteration or paraphrase of the intent.

Items that are not explicitly addressed by the standards, but are mentioned in C6 [1] or are merely included here for rational continuity of the document, are included in italics.

Title Page

The title page should be in the appropriate format for the organisation, with necessary identifications and approvals.

2.1 Purpose and Scope

State that the purpose of the document is to record the procedures established and implemented in the unit to manage all aspects of software quality assurance in the application and/or management and/or development of computer codes for safety analyses.

Add a brief, clear statement of the material covered in the document.

2.2 Definitions

The same approach taken in 1.7 above is recommended..

2.3 Applicable Reference Documents

The reference documents should include the CSA standards in 1.2 above and/or any other standards that the procedures are designed to follow.

2.4 Responsibilities and Unit Organisation

2.4.1 Mandate

This section should briefly outline the following attributes of the unit
- unit title
- identification of unit within the structure of the parent organisation
- unit mandate involving computer programs
- associated fields of specialisation
- customers and/or stakeholders

2.4.2 Responsibilities *and Roles*

This sub-section should define the following
- *organisation chart for the unit*
- *list of position titles, with necessary qualifications*
- definition of key roles, including for each role
  list of responsibilities
  minimum qualifications or training required

The standards state that the development and/or application of computer codes must be undertaken only by qualified personnel. *The USQAPD should specify minimum qualifications for each key role.*

Typical roles in software management are given in Appendix C, together with typical responsibilities. The key roles identified should be analogous to those defined in Appendix C, and responsibilities should be defined in similar terms.

2.5 Requirements for Computer Programs *according to Classification*

It is most unlikely that a computer code will be designed, written as planned, then verified, validated and issued without encountering a need for some change that was not anticipated in the original plan. Hence the need arises to develop and implement a version control system that defines a sequence of versions of a computer program that is under development.

Systematically documented control of a computer code is required to establish an unequivocal definition of all attributes of every version of the code, and a precise and traceable sequence of modifications from one version to the next. This also provides clear means of recovering previous versions by removing modifications. (The development and management of a code is a separate discipline from the application of a code. Application is discussed in section 2.10.)

This section of the USQAPD should define, for each type of computer code for which the unit is responsible, the level of control that the standards require. This involves interpretation of the specific requirements of section 5 of N286.7 for each computer code. *As the required level of control determines all the details of the necessary SQA procedures, the remainder of section 2 in this document contains recommended guidelines for establishing the classification of computer codes, and determining the requirements for each classification.*

2.5.1 Version identification

Version identification is an integral part of configuration management, a topic discussed in section 7 of N286.7. *However, the term is introduced here in this document, as much of the discussion in this section of the standards is best explained in terms of versions.*

A version of a computer program is a uniquely defined set of instructions, hence it may be argued , as in N286.7-96, that any change whatsoever to a computer program creates a new version, that should be identified as such.

In practical application it is preferable to use the word modification to identify an individual change, and reserve the term version to signify a particular stage of program development identified as a target or milestone in the development plan. Hence a number of separate or related modifications might be planned to convert the program from its current version to the next planned version.

The version identifier is a definitive part of the quality assurance records, *and should appear prominently in the output from the code, and in all documents referring to the code.*

There is no mandatory style for version identification, and a number of satisfactory schemes exist.

***The following example** uses a possible version identification scheme that has a hierarchy of several levels of identification as follows:*

CODENAME Vx,Ry,Mz
- *Vx: Version x - a change in version number x is planned whenever changes are made that require the data input format to change*
  - *this requires an associated revision of the user manual*
- *Ry: Revision y - a change in revision number is planned whenever changes are made that cause the output format to change*
- *Mz: Modification z - a new modification number is given to each modification task that is defined through the change control system*

The development plan could identify any VxRy as a target version that represents a particular stage in the development process. The targeted versions could be further identified as one of three levels

i) base version - a first prototype or reference version
ii) development version - having restricted access for testing and v&v purposes
iii) production version - being released for application

2.5.2 *Classification*

In this section, the USQAPD should include a catalogue of the computer programs that fall within the unit's mandate. The catalog should list each computer program that the unit applies for the purpose of safety analyses, or that the unit develops and/or modifies and/or manages for use in safety analyses done by other units. The catalog should classify each computer program according to the criteria discussed below.

Although the Q396 standards specifically address existing computer codes and codes under development, N286.7-94 concentrates more on the development life cycle involved in starting new computer programs, and mentions existing computer codes that are to be used either unchanged or with modifications as special cases. Most of the programs in use in the nuclear industry have a history of parallel development and application. It is extremely important that an appropriate level quality assurance should be applied to all such codes.

Recognising this, N286.7-96 has established more specific rules regarding management of this type of perpetual code development. It clear that such rules are essential to establish credibility in the application of this special category of codes.

For brevity, this document recommends the use of the term " type" to distinguish the above variations in computer code status, and that the USQAPD follow the intent of N286.7-96, by identifying each computer code that falls within the mandate of the unit as belonging to one of three types:

Type 1 : Computer Programs that were developed prior to the implementation of the standard addressed by the procedures, and that have not been changed.
Type 2: Computer Programs that were developed prior to the implementation of the standard addressed by the procedures, and that are undergoing any **significant change**.
Type 3: New Computer programs under Development

Significant change is defined in N286.7-96 as change in any of:
theoretical background
solution technique
calculation capability
data structure
embedded data in correlations
program structure
programming language
computer system assembler or compiler

The intended use of a particular computer program determines which CSA standard is applicable. *This document recommends that the term "category" be used to classify programs according to usage:*

Category	*Usage*	*Applicable Standard*
Category I	Software used in Critical Applications	CSAQ396.1
Category II	Software used in Safety Related Applications	CSA N286.7
Category III	Non-Critical Software	CSA Q396.2

The definitions of Critical, Non-Critical and Safety Related Usage are included in Appendix B, as given by the standards.

As most computer programs currently used in safety analysis are category II, type 2 , this document will focus primarily on this group, however other types and categories are discussed briefly.

2.5.3 Control of existing *(Type 1)* Software

In N286.7, older software, i.e. software that was developed prior to the issue of the standard and has not changed since, is specifically exempted from all of section 2.6 below, concerning program design and also from clause 2.11.2, that concerns documentation of program design. By implication, it is also exempted from clause 2.8 (change control) and much of 2.7 (configuration control). The AECB do not accept all these exemptions, however to satisfy the standard it is sufficient to identify the configuration of type 1 software by the following:

Program Name, Version and Version date
Program Abstract (including origin of program, hardware platform and system software)
User Manual
Theory Manual
Validation Report

2.5.4 Control of existing software requiring modifications *(Type2)*

If an existing computer program requires modification, it is first essential to clearly define a base version of the code before any modifications are initiated. This base version must be established and controlled as type 1 software and must have all the above attributes. The subsequent version which contains the planned changes will become type 2 software.

The standard states that all significant changes must then be planned and implemented in the same manner as new software under development *(type 3)*.

The USQAPD should require documentation of the attributes of the base version, and contain the procedures for planning and control of new developments.

2.5.5 Control of new software under development (Type 3)

New software must be developed in a controlled manner through every stage of the computer program life-cycle. The process is discussed in detail in N286.7-94 and -96. The following section retains the requirements of the standard, but attempts to condense these in a manner that is easier to follow. In summary the process consists of the following three components, each of which require associated documents.

Program Design, Development and Verification
Configuration Management and Change Control
Validation

These topics are discussed in the following sections:

2.6 Program Design and Development

The design and development process for a new computer program (or for significant changes to an existing program) should address and document each component of the following sequence:

i) Problem definition
ii) Requirements Specification
iii) Development Plan
iv) Theoretical Basis
v) Program design
vi) Coding
vii) Coding Review
viii) Code output requirements
ix) Verification
x) Validation
xi) Issue of the First Production Version and its associated Documentation
xii) Configuration management and change control

For new software *(type 3)* , the standards require advance planning such that the production, review and verification of detailed documents covering items i) to v) is completed before any coding is commenced. These documents will be substantial. The verification and validation stages should also be addressed in the planning document.

For existing programs requiring significant change *(type 2)* , the same components should be addressed, *however for modest changes, the documentation will be less bulky.*

The amount of detail required in this section is proportional to the magnitude of modification planned. *As a minimum the USQAPD should clearly state the sequence i)to xi), and provide minimum specifications for the document that will accompany each step of that sequence.* Following the topical structure of the standard, documentation is addressed specifically in section 2.11.

For type 3 software, the development plan may call for a series of target development versions that precede the first production version. If so, the first target version should be established as a base version and any ongoing changes should be implemented under configuration management.

If the development plan addresses only one target version, that first production version should be frozen as a base version and established as a type 1 program, and any further development not addressed in the original development plan would continue under configuration management and change control as type 2 modifications.

2.6.1 Coding Guidelines

Coding practice itself is not covered by an explicit part of the standard, however the standards do suggest some recommended practices. Appendix D has the list of recommended practices from N286.7, also reference [2] in the bibliography addresses coding practice.

Code output is not covered by an explicit part of the standard, *however it is recommended here that there be three mandatory components in the output of any controlled code:*

i) version identification should appear prominently on all output
ii) the code should print a condensation of the program abstract as an initial banner page
iii) all input data files should be reproduced verbatim in the initial output from the code.

2.7 Configuration Management and Change Control

Configuration management is the process of identifying configuration components, and maintaining the integrity and traceability of the arrangement of the components of a computer program. It comprises the sub-processes of:

i) **version identification**: identifying the configuration as a specific collection of computer program components that define a selected version of a particular computer program package operating on a given system as a specific product
ii) **version control**: maintaining the integrity of that product such that results from a given set of input data will always be reproducible.
iii) **change control**: ensuring that any changes to that product are implemented in a manner consistent with software quality assurance.

Change Control, item iii) above, ensures that:

a) changes are individually justified, planned, tested and documented
b) any subsequent versions of the product are clearly identified
c) any subsequent version can be systematically traced back to the previous version.
d) any previous version can be recovered
e) any previous results can be reproduced (if they were properly documented)

Item d) is a straightforward matter providing the computer system on which the program operates remains unchanged, however the standard requires consideration of bridging such changes in technology, and notes that any change in the host system software or hardware constitutes a new configuration that must be identified as such and linked to its predecessor.

2.7.1 Hardware Platform

The hardware and system software that are used for the development and validation of the computer program are an integral part of the configuration of a particular version. The porting of the code to a different hardware system constitutes a change in configuration and is subject to all the rules of change control and associated V&V.

It is recommended here that the configuration should be fully identified in the computer program abstract.

2.7.2 *Protection and Accessibility*

To ensure that the configuration integrity is protected, the code manager should implement system software controls on access to the source code and ensure that any modifications be planned and implemented through the change control system.

It is recommended here that:

a) users be permitted access only to object code of the designated production version.
b) developers be given read only access to the source code of the archived production version such that any modifications that they are developing and testing would generate a development version that remains in their own work area and causes no change to the production version. When modifications are complete they should be forwarded to the code manager for acceptance and implementation towards the next planned production version.

2.7.3 Code management software

The most effective way to manage code development is through the use of code management software. It is recommended here that the unit assembles or acquires a suite of code management software and uses this suite for the management and control of all software under care of the unit, thereby implementing a uniform approach to code management that is not dependent on the personnel involved with individual computing projects. These procedures themselves should be documented and controlled.

2.8 Change Control

2.8.1 Change control Planning

The motivation for changing a code may arise from two sources:

i) the need to enhance the code
ii) the need to correct a nonconformance or fault.

The development plans must address the first case and should also include provision to cater for the inevitable probability that corrections will be needed. The changes should be planned and documented using a standard change control form, *this form could cater for both the above needs.*

2.8.2 Change Control Implementation

Change Control Implementation and Testing should also be documented using a standard change control form.

2.8.3 Feedback

There should be regular interaction between the analysis teams and the code development and management teams. Any nonconformance experienced by the analysts should be reported *and the report formalised through the change control system; any requests for enhancement can be handled by the same route.* Clearly the code development and management teams should disseminate to all users any nonconformance reports and the plans to solve the problem, and plans for release of the next production version of the code.

2.8.4 Change Control Forms

The sample change control form given in Appendix E caters for all the above components of change control by using several sections.

2.8.5 Version Tracking

The standards require explicit documentation of the configuration of each version of a code and of each code change. This is normally done in a document referred to as a development history or version tracking record. This document should evolve concurrently with the development of each version, and a version of this document should be released at the same time as the code version is released.
A simple but efficient form of version tracking document consists of a compilation of completed change control forms.

2.9 Verification and Validation

As noted above, the required Verification and Validation (V&V) should be outlined in the original planning document, and also should be addressed in the change control documents for subsequent changes.

2.9.1 Verification

Verification is systematic confirmation that the coding itself and the results produced from selected basic test cases fulfill the specified requirements. *This can be done in a deterministic manner that does not require interpretation, so the specification of verification tests is straightforward, and the result is normally an unequivocal pass or fail.*

2.9.2 Validation

Validation is required to provide sufficient information to justify the use of the computer code in a particular application. *Such justification involves quantifying the uncertainties associated with the key results sought in that application and demonstrating that that magnitude of uncertainty is acceptable.*

The validation process involves comparison of results produced by the particular code against whatever relevant evidence can be found. Such evidence may include data from operating sites, related experiments, standard benchmark problems or from other computer codes that have been validated for the results considered in the comparison. *Since the amount of reference data available is always limited, has some associated uncertainty and also may not directly pertain to the application that the validation is intended to justify, the validation process cannot be expressed in such unequivocal terms as verification and an extensive validation program consisting of a repertoire of comparisons that examine different aspects of the results is usually necessary.*

The most meaningful approach to validation is to concentrate on the validity and uncertainty of computed key parameters that are crucial to the conclusions of each analysis planned. To this end, the AECB and the industry have launched an extensive initiative on validation that identifies these key parameters for each licensing scenario and focuses on their associated uncertainties. This is a separate initiative and will not be further addressed in the current document.

2.10 Application of Computer Codes

As the development, management and application of a computer code are each distinct disciplines, *it is useful to regard user groups as the end customers of the code development team and the code management team.*

Given this relationship, the code management team must provide a properly controlled configuration with all associated documentation for use by the application group.

2.10.1 Analyses

The application group must ensure that all components of the analysis are defined in a manner that permits the results they generate to be reproducible, reasonable, and easily reviewed. To do this they should document in advance:

- *the goal and scope of the analysis,*
- *the specifications of the physical system to be modelled*
- the assumptions involved in generating the simplified model to be used to represent the physical system
- *a quantitative assessment of the degree of conservatism inherent in the assumptions*
- assembly, referencing and certification of data describing that model
- the choice of computer code to be used
- the choice of a configuration of the code that is fully identified, *and documented as a production version*
- quantitative evidence that this configuration has been adequately validated for the intended use
- the assembly of input data and verification that it adequately represents the system
- evidence that the code will be used only within it's documented range of applicability and in a manner consistent with it's documentation
- *a plan to present results that clearly demonstrate that the goal of the analysis has been attained*
- a plan to assess results to confirm that they are appropriate
- *a methodology to assess the uncertainty of the key results, including those uncertainties involved in extrapolating when necessary from the assembled validation data to the application in question.*

Clearly, any analysis requires a final report that details all the above, and presents results, associated uncertainties and interprets these in the context of the goal of the analysis. However interpretation of results is beyond the scope of the CSA SQA standards, and hence need not be addressed in the USQAPD.

2.10.2 *Validation Analysis*

Any validation analysis is an application and should be planned and documented as in section 2.10.1.

2.10.3 Feedback

Any nonconformance or questionable performance by the code that is encountered by the analysts should be clearly defined and reported to the code manager.

2.10.4 Computer Program Transfer

If a particular version of a code is physically or electronically transferred from the code manager to a user group, the user group development must confirm that the version thus obtained passes verification tests specified by the code manager, prior to using that version for other analyses.

2.10.5 *Standard Data Decks*

For any suite of analyses to be done for a given physical system, it is recommended that a standard data deck be prepared and verified, and then maintained under the change control process, so that the initial reference deck and all subsequent changes are verifiable and retrievable.

2.11 Documentation

The standards require specific documents to accompany each of the activities discussed above, and outline details of the expected contents of each. To aid the specification of documents in the USQAPD, the required documentation is summarised in Appendix F.

2.12 *Implementation and Revision of Procedures, SQA Audits*

This topic is not addressed in N286.7, but is required in Q397, and AECB-C6. Having established the procedures under the above topics (2.1 - 2.11), the final section of the USQAPD should address the following:

i) implementation schedule
ii) provision of sufficient records to demonstrate in an auditable manner that the procedures have been implemented and are being followed
iii) schedule for regular review and possible revision of the procedure document
iv) person responsible for i) to iii) above.

2.13 *Bibliography*

The term reference in section 2.3 implies reference standards, hence the bibliography should be contain any other type of report that the USQAPD refers to.

2.14 Appendices

Appendices should be used for appropriate support documentation..

3. Sample USQAPD (Unit Software Quality Assurance Procedures Document)

This section contains a sample outline for a procedures document for a fictional unit "xxx" in a fictional organisation "yyy'. The sample has been created by applying the guidelines of section 2, and the subsections follow those of section 2 precisely. The sample can be made to look like a self-contained document in which the sections are equivalent to those of N286.7, by merely removing the prefix "3." from each subsection of section 3.

Title Page (sample)

Procedures for Software Quality Assurance
in
The xxx unit

yyy Organisation

Report No..... Revision No..... Date.....
Written by ..... Date .....
Approved by ..... Date.....

Full organisational Identification

3.1 Purpose and Scope

The purpose of this procedure document is to record the procedures established and implemented in the unit to manage all aspects of software quality assurance in the development, management and application of computer codes for safety analyses.

The document delineates specific procedures for all aspects of computer code work in the xxx unit, and specifies formats for all documents associated with such work.

3.2 Definitions of Terms used in this Document

Appendix A lists definitions of terms used in this document. It contains all the definitions from CSA.N286.7-94 plus some additional definitions that are included to clarify the current document.

3.3 Applicable Reference Documents

The Canadian Standard Association series of N286 documents define quality assurance standards for activities related to a nuclear power plant project.

CAN/CSA N286.0-92 Covers Overall Quality Assurance
CAN/CSA N286.2-86 Covers Design Quality Assurance

CSA N286.7-94 Covers Software Quality Assurance

CSA N286.7-94 has been issued as a preliminary standard entitled: Quality Assurance of Analytical, Scientific and Design Computer Programs for Nuclear Power Plants. The standard addresses computer programs for analytical, scientific and design applications for safety-related systems and components, and applies to programs originating in the nuclear industry and programs procured from external vendors. This standard has been under development for several years, and is likely to be further revised following considerable interaction with the industry.

A draft revision dated March 1996 has also been consulted in preparing this document, this revision is referred to as N286.7-96.

The CAN/CSA Q396 series (1989) has four documents that address SQA programs for i) the development of Software and ii) the use of Previously developed software in a) Critical Applications and b) Non-Critical Applications. These standards cover many of the topics included in N286.7, but concentrate more on the SQA aspects of the customer client relationships.

This document refers to CSA N286.7 and the CSA Q396 series collectively as the CSA SQA standards.

3.4 Unit Organisation and Responsibilities

3.4.1 Mandate

The xxx unit reports to the zzz division in the yyy organisation.

Its mandate includes the management of the AAA and BBB computer codes, validation of the codes, improvements to the codes and interaction with users. Some application of the codes, primarily to validation is done within the unit, the codes are also applied to safety analyses by other user groups at various sites.

These computer codes are primarily thermalhydraulic codes but include some reactor physics and fuel modelling. Further details are given in section 3.5.

The principal customer is the zzz division of the yyy organisation. The AECB could be regarded as an ultimate customer.

3.4.2 Roles and Responsibilities

This section defines the key roles and responsibilities within the unit.

The Unit Manager
is supervisor of all the unit's activities, and has the following responsibilities:
- accepts projects, appoints project leaders
- reviews/ approves project plans
- assigns qualified support staff and appropriate resources
- implements training when necessary
- reviews/approves all documents
- approves SQA procedures, and monitors their implementation and review
- minimum qualifications - post graduate degree in science or engineering or equivalent, plus six years experience in computational analysis in thermalhydraulic systems

Two Project Leaders, the Code Manager and the Senior Applications Analyst, report directly to the Unit Manager.

The Code Manager
- reports to the unit manager
- is project leader in charge of controlling the computer codes AAA and BBB
- directs and monitors configuration management and change control
- formulates SQA plans for code enhancements
- defines verification/ validation requirements
- supervises and directs the code custodian and one development analyst
- defines, assigns and coordinates tasks
- reviews and accepts tasks on completion
- responsible for all documentation related to the code
- minimum qualifications - post graduate degree in science or engineering or equivalent, plus
  four years experience in code development in thermalhydraulic systems

The Code Custodian
- reports to the code manager
- assembles official versions of the computer code
- implements and maintains these versions on the hardware platform
- implements configuration management, change control and verification
- archives code back up versions, and all modifications
- may initiate change requests
- arranges workshops for new staff on code use
- minimum qualifications - technical diploma in computer science or equivalent, plus
  two years experience in computational analysis

Development Analysts
- report to the code manager
- responsible for completing particular code development tasks
- may initiate change requests
- minimum qualifications - undergraduate degree in science or engineering or equivalent, plus
  two years experience in computational analysis in thermalhydraulic systems
  attended one or more code workshops

Senior Application Analyst
- reports to the unit manager
- is project leader in charge of all application of computer codes in the unit
- formulates SQA plan for each analysis project
- defines verification/ validation requirements
- supervises and directs the two application analysts
- defines, assigns and coordinates tasks
- reviews and accepts tasks on completion
- responsible for all documentation related to the application
- reports code problems to code manager
- may initiate change requests
- minimum qualifications - post graduate degree in science or engineering or equivalent, plus
  four years experience in computational analysis in thermalhydraulic systems
  attended one or more code workshops

Application Analysts
- report to the senior analyst
- responsible for completing and verifying code application tasks assigned by the Senior Analyst
- reports analysis problems to project leader
- minimum qualifications - undergraduate degree in science or engineering or equivalent, plus
two years experience in computational analysis in thermalhydraulic systems or
attended one or more code workshops

Independent Reviewer
- a part time responsibility assigned as needed to any of the above personnel
- appointed by and reports to the unit manager, code manager or senior analyst to review particular task(s)
that have been completed by other analysts
- minimum qualifications - post graduate degree in science or engineering or equivalent, plus
two years experience in computational analysis in thermalhydraulic systems

SQA Representative
- a QA specialist assigned by the organisation on an as needed basis
- assists with or reviews the unit's SQA procedures
- minimum qualifications - post graduate degree in science or engineering or equivalent, plus
two years experience in SQA

3.5 Computer Program Classification and Requirements for Control

3.5.1 Version Identification

Version identification for the computer codes managed in the unit follows the convention Vx,Ry,Mz, i.e. Version x, Revision y, Modification z.

The numbers x and z change monotically. The modification number, z is incremented each time the code is changed through the change control procedure, and is in fact identical to the task number of the latest modification. This modification number uniquely identifies the content of the code. The version number x, and the Revision number y are used to indicate whether modification level z constitutes a frozen production version that is approved and released for use, or is still undergoing test and/or development.

Each production version is released identified as Vx,R0,Mz , i.e. Version x, Revision 0, (Modification z).

Continuing development through the change control process requires that the code content at the next change be identified Version x, Revision 1, Modification z+1.

Each subsequent coding change, no matter how small is done through the change control process and requires an increment in the modification number , the version and revision numbers are assigned to change according to target projects in the development plan.

The version identifier, Vx,Ry,Mz, appears on each page of code output to uniquely identify the content of the code that produced that output.

3.5.2 Classification

Following the intent of CSA.N286.7, the following basis for classification is used:

Type 1 : Computer Programs that were developed prior to the implementation of the standard addressed by the procedures, and that have not been changed.
Type 2: Computer Programs that were developed prior to the implementation of the standard addressed by the procedures, and that are undergoing any significant change.
Type 3: New Computer programs under Development

The computer programs are also classified according to usage category:

Category	Usage	Applicable Standard
Category I	Software used in Critical Applications	CSAQ396.1
Category II	Software used in Safety Related Applications	CSA N286.7
Category III	Non-Critical Software	CSA Q396.2

Given the above, the computer codes managed by the unit are summarised in table 1:

Table 1
Classification and Description of Computer Codes managed by the xxx Unit

Code	Classification	Purpose and Use	Current Version & Date	Current Documentation
AAA	Category II Type1	Steady-state thermalhydraulic code used for scoping and reference calculations	V12 92/07	Code abstract xxx-92-12 User Manual xxx-92-17 Theory Manual xxx-92-22 Valid'n Report xxx-93-32
BBB	Category II Type2	Transient two-fluid system thermalhydraulics code with coupled fuel thermal and physics	V5R0 (M122) 96/02	Code Abstract xxx-95-25 User manual xxx-95-27 Theory Manual xxx-95-33 Developm Plan xxx-95-51

3.5.3 Control of existing (Type 1) Software

Each production version is frozen at issue and hence becomes type 1 software. Each issue is accompanied by commensurate revision of the following documentation:

Program Abstract, including hardware platform and system software
User Manual
Theory Manual
Validation Report

3.5.4 Control of existing software requiring modifications (Type2)

As noted above, any change to the current production version Vx,R0,Mz requires that the resulting code must be identified as Vx,Ry,Mz', where y>0 and z'>z.

Changes are planned, implemented, tested and accepted through the change control process described in section 3.8.

Each document contains the Program Name, Version Identifier and release date, and identifies the differences found in the new production version and the planning rationale behind those differences.

The Program Abstract is reissued in full with each release, however incremental revisions to the theory manual, user manual and validation report are sufficient when the changes are not extensive.

3.5.5 Control of new software under development (Type 3)

The current mandate of the unit does not include projects involving development of new software.

3.6 Program Design and Development

As the unit has no new software projects, the planning of changes is handled through the change control process. That process does incorporate in brief the following planning sequence that is required by the standard for the development of new software:

i) Problem definition
ii) Requirements Specification
iii) Development Plan
iv) Theoretical Basis
v) Program design
vi) Coding
vii) Coding Review
viii) Code output
ix) Verification Tests
x) Validation
xi) Criteria for selection and Release of Production Version.

3.6.1 Coding guidelines

Coding practice follows the list of recommended practices from N286.7, given in Appendix D.

The version identifier, Vx,Ry,Mz, appears on each page of code output to uniquely identify the content of the code that produced that output.

The codes issue a banner page that reproduces the program abstract.

All input data files are reproduced verbatim in the initial output from the code.

3.7 Configuration Management and Change Control Process

3.7.1 Configuration identification & hardware platform

The configuration for each code is fully defined in the program abstract. Any change in the hardware platform and/or system software will require a new version to be issued.

3.7.2 Protection and Accessibility

To ensure that the configuration integrity is protected, the code manager has implemented system software controls on access to the source code.

Users are permitted access only to object code of the designated production version.

Developers are given read only access to the source code of the archived production version such that any modifications that they are developing and testing generates a development version that remains in their own work area and causes no change to the production version.

When modifications are complete they are forwarded to the code manager for acceptance and implementation towards the next planned production version

3.7.3 Configuration management software

The code manager uses the UNIX Revision Control System (RCS) for managing all code revisions.

3.8 Change Control

3.8.1 Change Control Planning

The codes develop in an evolutionary manner primarily driven by interaction with our customers.

The motivation for changing a code may arise from two sources:

i) the need to enhance the code
ii) the need to correct a nonconformance or fault.

Changes needed to enhance the code are identified in the code development planning document, reference {1}. The ongoing development is planned to address a series of goals. A new version of the code will be released on the attainment of each major goal.

For each goal the document outlines the following:

Overall Technical Requirements to address the goal
Specific Tasks required to meet these requirements
Individual modification sets required to accomplish these task
(a modification set in the change process consists of a group of related code changes that can be planned, implemented and tested as a unit).
Verification tests required for acceptance of modification
Verification tests required for acceptance of task as complete
Verification tests required for acceptance of goal as complete in principle
Validation tests required for acceptance of goal as complete in intent

3.8.2 Change Control Process

Once the planning process for the next release version is complete, the individual modifications required are recorded in a Change Request Form. The actual form used is given in Appendix D
This form combines all information needed to complete the following :

A. Change Request - Defined by Request Originator
- Request originator & Date
- version id. that requires change
- reason for request
- features that need changing
- acceptance criteria for completion of change

B. Implementation Plan - Defined by Code Manager
- reason if request not accepted
- if accepted
  - request number assigned
  - theoretical/numerical background if applicable
  - summary of coding required
  - routines affected
  - assigned to ...., for completion by ....
  - scheduled to be complete for production version Vx (Anticipated date ....)
  - benchmark problems to be tested against
  - expected effect on results
  - acceptance criteria
  - updates to manuals if required
  - coding and results to be checked by ....
- feedback copy of above sent to originator

C. Implementation in Development Version
- modification accepted by code manager
- assigned modification number z
- integrated in development version
- integration tested and accepted
- modification archived
- documents revised if necessary
- feedback report to development team and users

3.8.3 Version Tracking

Once received by the code manager, this change control form resides in a change request folder until the completion of all the above.

The completed form is then archived in the Code Development History Document. This document is not published formally as it expands with each change, but it will be maintained on disc in electronic form for access by developers and customers.

3.8.4 Feedback

Formal means of feedback is maintained through the change control system.

3.9 Verification and Validation

3.9.1 Verification

The required verification for changes is included in the change control process.

3.9.2 Validation

Since both computer codes under management by the unit are existing codes, a number of validation documents already exist. These are a good measure of the credibility of the code under certain circumstances. However further enhancement of the codes is deemed necessary to satisfactorily complete the validation process for all licensing scenarios. The validation required necessarily increases with the evolution of the codes and their mandate. Hence a more comprehensive validation plan has recently been developed, reference {2}. This plan follows the extensive initiative that the industry and the AECB have recently launched on validation, and identifies the key parameters for each licensing scenario, and focuses on their associated uncertainties. This initiative is not discussed further in the current document.

This validation plan is currently the driving force in the development plan.

3.10 Application of Computer Codes

The Senior Analyst is given charge of each set of analyses, writes an analysis plan, and assigns and supervises tasks. He requires access to the current production version of the relevant computer program, and is a customer of the code manager and his team.

Given this relationship, the code management team must provide a properly controlled configuration with all associated documentation for use by the project leader's application team.

3.10.1 Analyses

The senior analyst must ensure that all components of the analysis are defined in a manner that permits the results they generate to be reproducible , reasonable, and easily reviewed.

This is done by documenting in advance:
- the scope, and goal of the analysis,
- task breakdown
- the specifications of the physical system to be modelled
- the assumptions involved in generating the simplified model to be used to represent the physical system
- assembly, referencing and certification of data describing that model
- the choice of computer code to be used
- the choice of a configuration of the code that is fully identified, and documented as a production version
- evidence that this configuration has been validated for the intended use
- the assembly of input data and verification that it adequately represents the system
- evidence that the code will be used only within it's documented range of validation and in a manner consistent with it's documentation
- a plan to present results that clearly demonstrate that the goal of the analysis has been attained
- a plan to assess results to confirm that they are appropriate
- a methodology to assess the uncertainty of the key results
- review by independent reviewer and subsequent acceptance by project leader
- archive of results and data deck

The applications analyst conducts particular analysis tasks. His work is supervised closely by the Senior Analyst, and is reviewed by an independent reviewer and the Senior Analyst.

The analysis planning and implementation form given in Appendix E is used as a checklist for the entire process.

Each analysis requires a final report that details all the above, and presents results, associated uncertainties and interprets these in the context of the goal of the analysis. However interpretation of results is beyond the scope of the standards, and hence is not addressed here.

3.10.2 Validation Analysis

Each new validation analysis is an application and is planned and documented as in section 9.1

3.10.3 Feedback

The code manager is responsible for maintaining regular contact with all application teams. Any nonconformance experienced by the analysts is reported through the change control system, and any requests for enhancement are handled by the same route. The code manager uses the change control system to disseminate to all users any nonconformance report and the plans to solve the problem, and plans for release of the next production version of the code.

3.10.4 Standard Data Decks

A standard data deck is prepared as the first task in any suite of analyses for a given physical system. This is established as a reference and maintained under the change control process, so that the initial reference deck and all subsequent changes are verifiable and retrievable.

3.11 Documentation

Specific documents accompany each of the activities discussed above, and details of the expected contents of each are outlined in the CSA standards. Required documentation is summarised in Appendix F.

3.12 Implementation and Audits

3.12.1 Implementation

Implementation of the procedures described above was commenced in April 1996, all work now adheres to these procedures.

3.12.2 SQA Records

The procedures require the use of the forms in Appendix E. All key stages of work are archived, both in code management and in application. These forms are also filed in hardcopy and contain the path to the appropriate archived computer files.

3.12.3 Review of these Procedures

The unit manager is responsible for initiating review of these procedures. The procedures will be reviewed on an annual basis, the next review being April 1997.

3.13 Bibliography

{1} Development Plan for the BBB Computer Program, xxx-95-51.
{2} Validation Plan for the BBB Computer Program, xxx-96-02

4. Concluding Remarks

This report has provided guidelines for writing a set of SQA procedures that satisfy the N286.7 standard, and has used those guidelines to present a sample SQA procedures document for a fictional organisational unit. A number of sample SQA forms have been included to show that the requirements of the standards can be met by relatively concise records.

5. Bibliography

1. Requirements for the Safety Analysis of CANDU Nuclear Power Plants, AECB Consultative Document C-6,R1,1994.

2. Guidelines for the Development and Use of computer Programs, APEO, 1987.

Appendix A
Relationship of Procedures Document to CSA Standards

	CSAN286.7-94	CSAN286.7-96	CSAQ396.1.1 & .2 CSAQ396.2.1 & .2
Title Page	not mentioned		
1 Purpose and Scope	1 Scope		
2 Definitions	2 Definitions		
3 Applicable Reference Documents	3 References		
4 Unit Organization and Responsibilities	4 Responsibilities		
5 Computer Program Classification and Requirements for Control	no specific section	5 Requirements for Computer Programs	(covered by 396.1.1 or .1.2 or 396.2.1 or .2.2)
6 Computer Program Design and Development	5 Computer program Design & development	6 Computer program Design & development	5.6 Quality Project Plan
7 Configuration Management	6 Configuration Management 8 Hardware Integration	7 Configuration Management	6.3 Configuration Management & Control
8 Change Control	7 Change Control	8 Change Control	6.3
9 Verification and Validation	5.5.2 Verification 5.6 Validation	6.7 Verification 9 Validation	5.6.6 Verification 5.6.7 Validation
10 Application of Computer Codes	10 Execution of Computer Programs	10 Use of Computer Programs	not specific
11 Documentation	10 Documentation	11 Documentation	5.6 Quality Project Plan
12 Implementation, Revision & Audits	not specific		5.4 SQA Review and Audit
13 Bibliography	not specific		

APPENDIX B - DEFINITIONS

All the definitions in CSA.N286.7-94 are reproduced here in standard font, some additional definitions from CSA.N286.7-96(draft) and CSA.Q396 have been added where they are considered useful for the current document and are marked appropriately. *Some additions have been introduced solely for the purpose of the current document ; these appear in italic font.*

Acceptance testing - formal testing conducted to determine whether or not a system satisfies the acceptance criteria as set out by the owner or participant and to enable the customer to determine whether or not to accept the system.

Algorithm - a finite series of well defined rules
(a) for the solution of a problem in a finite number of steps.
(b) that gives a sequence of operations for performing a specific task.

Archiving - a formal process for storage of files in a manner that ensures complete retrieval..

Assemble- to translate a program expressed in an assembly language into a machine language.

Assembly language - a machine-specific language containing instructions that are usually in one-to-one correspondence with computer instructions.

Base Version - *the first version of a computer program that is put under configuration control .*

Central processor unit - computing system that contains the circuits that control the interpretation and execution of instructions, including the necessary arithmetic, logic, and control circuits to execute the instructions.

Change control - the process by which a change is proposed, evaluated, approved or rejected, scheduled, and tracked.

Coding - translating a computer program design into a programming language.

Coding Project - *a defined project requiring coding*

Coding Task - *a defined subordinate part of a computer project*

Comment Statement - a statement that is embedded in a computer program in order to provide clarification to human readers and that does not affect machine interpretation.

Compiler - a program to translate a high level language program into its machine code equivalent.

Computer Code or Computer program - a sequence of instructions suitable for processing by a computer and designed to achieve a certain result.

Computer program abstract - a brief description of a computer program, providing sufficient information for potential users to determine the appropriateness of the computer program to their needs and resources.

Computer program routine - a computer program segment that performs a specific task.

Computer system - a system composed of computer(s), peripheral equipment such as disks, printers and terminals, and the computer programs necessary to make them operate together.

Configuration - the precise combination of software components that are linked to operate *as a specific product* as set forth in the accompanying documentation (CSA.Q396).

Configuration component - a collection of computer program elements treated as a unit for the purpose of configuration management.

Configuration management - the process of identifying configuration components, controlling changes, and maintaining the integrity and traceability of the arrangement of the computer program components.

Construct - a complex image or idea formed from a number of simpler images or ideas.

Control structure - a construct that determines the flow of control through a computer program.

Correctness - the extent to which a computer program is free from design and coding defects.

Critical Application - the use of software in which the occurrence of failure may have an effect on health and/or safety, or may cause significant financial or social loss.

> **Non-critical application** - the use of software in which the occurrence of failure is unlikely to cause any of the above effects.

Data structure - a representation of logical relationships among individual data elements.

Debugging - the process of locating, analysing, and correcting suspected faults.

Design phase - the period of time in the computer program life cycle during which the designs for architecture, computer program components, interfaces, and data are created, documented, and verified to satisfy requirements.

Developer - the party that creates a computer program or a component thereof and its associated documents for its own use or that of others.

Development phase - the period of time that begins with the decision to develop a computer program and ends when the computer program is delivered.

Document - a data medium and the data recorded on it, that generally has permanence and that can be read by a person or machine.

Documentation - a collection of documents on a given subject.

Embedded computer program - a computer program for an embedded computer system.

Embedded computer system - a computer system that is integral to a large system whose primary purpose is not computational.

Empirical correlation - a mathematical representation of an assumed interdependence between two or more variables based on actual measurement, observation or experience, rather than theory.

Error - a discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition, including that resulting from human actions.

Executable code - a computer program in a language that can be directly executed by a computer.

Fault - a manifestation of an error in a computer program.

File - a set of related computer records treated as a unit.

Flowchart - a graphical representation of the definition, analysis, or solution of a problem in which symbols are used to represent operations, data, flow, and equipment.

Hardware - physical equipment used in data processing, as opposed to computer programs, procedures, rules, and associated documents.

Hierarchy - a structure whose components are ranked into levels of subordination according to a specific set of rules.

High level language - a programming language that usually includes features such as nested expressions, user defined data types, and parameter passing not normally found in lower level languages, that does not reflect the structure of any one given computer or class of computers and that can be used to write machine independent source programs.

Instruction - an element of a computer program containing a meaningful expression that causes a computer to perform a particular operation

Integration - the process of combining computer program elements, hardware elements, or both into an overall system.

Integration testing - an orderly progression of testing in which computer program elements, hardware elements, or both are combined and exercised until the entire system has been integrated.

Interface - a shared boundary between computer system components.

Interpret - to translate and to execute each source language statement of a computer program before translating and executing the next statement.

Job control language - a language that expresses the statements of a job and that is used to identify the job or describe its requirements, usually to an operating system.

Latent defect - a fault which exists but is as yet concealed.

Life cycle - the period of time that starts when a computer program is conceived and ends when the program is no longer available for use and analyses performed with it no longer support any nuclear power plant operating licenses.

Mathematical Library - a collection of computer program modules that perform standard mathematical operations (CSA.N286.7 - 96 (draft))

Milestone - a scheduled event that is used to measure progress.

Model - a representation of a device, concept or a physical process.

Modification - *i)any change to software*
ii) a set of related changes to software made to complete a particular coding task

Module - a computer program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading into the computer.

Module header - a set of comments forming part of the source code of a module identifying it, and defining its purpose and use.

Nest - to incorporate a structure or structures of some kind into a structure of the same kind.

Nonconformance - a deficiency in characteristic, documentation, or procedure that renders the quality of a computer program unacceptable or indeterminate, or not according to specified requirements.

Object module - a module containing a fully compiled or assembled program that is ready to be loaded into a computer.

Operand - an entity on which an operation is performed.

Operating system - computer programs that control the execution of computer programs, provide hardware resource allocations, input and output control, and related services.

Organization - the owner of or participant in a nuclear power plant project.

Owner - the party who has or will have title to a nuclear power plant.

Parameter - a variable that is used to pass values between program routines.

Participant - an organization required by the owner to meet one or more of the second tier standards in the N286 series.

Platform *- the computer system (hardware and software) on which a computer program has been implemented and tested.*

Portability - the ease with which a computer program can be transferred from one computer system or environment to another.

Process - in a computer system, a unique, finite course of events defined by its purpose or by its effect, achieved under given conditions.

Program Component - a basic part of a computer program.

Programming language - an artificial language designed to generate or express computer programs.

Program loop - a set of instructions in a computer program that may be executed repeatedly while a certain condition prevails.

Programming segment - the sequence of computer program statements between two consecutive branch points.

Program statement - see source statement.

Program structure - a representation of the logical relationship among individual program components.

Qualification - the process of determining whether a system or component of a system is acceptable for use (IEEE Std 610.12-1990)

Quality assurance - a planned and systematic pattern of actions designed to provide adequate confidence that a computer program will be of the required quality.

Qualified person *- a person who possess an appropriate level of expertise and experience to perform the assigned task.*

Record - information stored in a document or other media that furnishes evidence of the quality and/or activities.

Safety-related systems - those systems, and the components and structures thereof, which, by virtue of failure to perform in accordance with the design intent, have the potential to impact on the radiological safety of the public or plant personnel from the operation of the nuclear power plant. Those systems, and the components and structures thereof, are associated with
(a) the regulation (including controlled startup and shutdown) and cooling of the reactor core under normal conditions (including all normal operating and shutdown conditions);
(b) the regulation, shutdown, and cooling of the reactor core under anticipated transient conditions, accident conditions, and the maintenance of the reactor core in a safe shutdown state for an extended period following such conditions; and
(c) limiting the release of radioactive material and the exposure of plant personnel and/or the public to meet the criteria established by the licensing authority with respect to radiation exposure during and following normal, anticipated transient conditions and accident conditions.
Notes:
(1) The term "safety-related system" covers a broad range of systems, from those having very important safety functions to those with a less direct effect on safety. The larger the potential radiological safety effect due to system failure, the stronger the "safety-related" connotation.
(2) The term "safety-related" also applies to certain activities associated with the design, manufacture, construction, commissioning, and operation of safety-related systems, and to other activities which could similarly affect the radiological safety of the public or plant personnel, such as environmental and effluent monitoring, radiation protection and dosimetry, and radioactive material handling (including waste management). The larger the potential radiological safety effect associated with the performance of the activity, the stronger the "safety-related" connotation.
(3) Certain failures of other systems could adversely affect a safety-related system (e.g. through flooding or by mechanical damage). The potential for this and the necessary means to control it must be addressed in appropriate phases.

Software - a program or set of programs and associated data, procedures, rules, documentation and materials concerned with the use, operation and maintenance of a computer system or an automated information system. CSA.Q396

Software Component - a functionally and logically distinct part of a software program CSA.Q396

Software Product - the complete set of computer programs, procedures, associated documentation and data designated for delivery to the user (IEEE Std 610.12-1990)

Software Quality Assurance (SQA) - a planned and systematic pattern of all actions necessary to provide adequate confidence that software components conform to established requirements and specifications. CSA.Q396

SQA Audit - a formal, planned and documented activity that will objectively evaluate and verify that the applicable elements of an SQA program have been established, documented and implemented in accordance with the applicable standard. CSA.Q396

SQA Procedure *- a document prepared by a functional unit within an organisation that specifies and assigns all activities that are required to establish an acceptable SQA program to govern the software activities of that unit.*

Source Code - a computer program that must be compiled, assembled, or interpreted before being executed by a computer.

Source language - a language used to write programs that must be compiled, assembled, or interpreted before being executed by a computer.

Source program - a computer program written in source language.

Source statement - an instruction in a source program

Subprogram - a computer program unit that may be invoked by one or more other computer program units.

System - a collection of hardware, computer programs, people, facilities and procedures organised to achieve a common objective.

System Library - a controlled collection of programs, resident on the computer system, that can be accessed for use or incorporated into other programs by reference.

Testing - the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements.

Unit - *(in this document) a subgroup (or team) within an organisation.*

User - a party which uses a computer program to perform analytical, scientific and/or design computations related to nuclear power plant systems, equipment and components.

Validation - comparison of the results of the computer program with measurements, experiments or known solutions, so that the accuracy or uncertainty for a particular application can be determined.

Variable -
(a) a quantity that can assume any of a given set of values;
(b) a character or a group of characters that refers to a value and, in execution of a computer program, corresponds to a specific memory location.

Verification - the process of determining whether or not the products of a given phase of the computer program development cycle fulfill the requirements established during the previous phase. Reviews, testing and walk-through are some of the means of accomplishing verification.

Note: This definition of the term "Verification" applies to activities related to the computer program development cycle and differs from the definition given in CSA CAN3-N286.2 which applies to design activities.

Version - *A term used to identify a software product having a uniquely defined composition*

Walk-through - a review process in which a designer or programmer leads one or more other members of the development team through a segment of design or computer program that she/he has written, while the other members ask questions and make comments about technique, style, possible errors, violation of development standards, and other problems.

Appendix C
Typical Roles & Responsibilities

Unit Manager
- senior supervisor of the unit
- accepts projects, appoints project leaders
- reviews/ approves project plans
- assigns qualified support staff and appropriate resources
- reviews/approves all documents
- approves and implements SQA procedures
- implements training programs for new staff

Project Leader
- in charge of particular software project(s)
- defines and coordinates tasks
- formulates SQA plan for project
- reviews and accepts tasks on completion
- responsible for all project documentation

Code Manager
- project leader in charge of controlling a particular computer code
- directs and monitors configuration management and change control
- defines verification/ validation requirements
- responsible for all documentation related to the code
- directs code custodian and development analysts

Code Custodian
- assembles official versions of the computer code
- implements and maintains these versions on the hardware platform
- implements configuration management, change control and verification
- reports problems to code manager
- may initiate change requests

Development Analyst
- responsible for completing particular code development task(s)
- reports problems to code manager
- may initiate change requests

Application Analyst
- responsible for completing particular code application task(s)
- reports code problems to code manager
- reports analysis problems to project leader
- may initiate change requests

Independent Reviewer
- may hold one of the above positions
- appointed by manager, code manager or project leader to review particular task(s) that have been completed by other analysts

SQA Representative
- a QA specialist assigned by the organisation to assist with or review the unit's SQA procedures

Appendix D
Recommended Checklist to Confirm Programming Practice (from CSA N286.7-94)

Note: *This appendix is not a mandatory* part of the Standard

Documents defining standard and uniform programming practices, such as programmers' handbooks, should address the following issues:

(a) Program Organization
  (i) Overall program structure, including
    (1) input data acquisition and processing;
    (2) checks and edits;
    (3) computation and data processing; and
    (4) final edits and saving of specific data.
  (ii) orderly progression of calculational flow;
  (iii) overall program control in a single module;
  (iv) a single well defined function for each module;
  (v) a uniform module layout, including
    (1) module header;
    (2) data definition; and
    (3) programming segments.

(b) Programming Language
  (i) specification of standard programming language;
  (ii) adherence to an applicable programming standard;
  (iii) limited use and thorough documentation of extensions to the standard programming language;
  (iv) use of assembly language only where absolutely necessary and thorough documentation of the programming logic.

(c) Data Management
  (i) documentation of data transfer techniques;
  (ii) use of one data transfer technique throughout the computer program unless a data structure requires special handling;
  (iii) use of one module to read or write a data file;
  (iv) documentation of data file content and structure.

(d) Computer Program Features
  (i) input data identification, preparation, organization;
  (ii) initialization of arrays, variables, default parameters;
  (iii) reproduction of input data for visual inspection;
  (iv) input and output error checking, including array bounds;
  (v) testing of intermediate results;
  (vi) reports about calculational path, intermediate results, and calculation progress;
  (vii) adequacy of error-checking and associated warning and error messages;
  (viii) normal and abnormal computer program terminations;
  (ix) output information formats and users' options.

(e) Source Statements and Variables
  (i) specification statements for variables;
  (ii) meaningful variable names;
  (iii) use of comment statements;
  (iv) reflection of program structure and techniques in comment statements;
  (v) unique identification of comment and source statements;
  (vi) single purpose use of variables;
  (vii) checking of parameter types;
  (viii) arrays of fixed length.

Appendix D
Recommended Checklist to Confirm Programming Practice (from CSA N286.7-94)
(Continued)

(f) hardware considerations
  (i) independence from unique hardware or computer program features;
  (ii) documentation of computer memory management during computer program execution;
  (iii) impact of computer memory management on computer program portability.
(g) Good Programming Techniques
  (i) selection of variable names to reflect scientific notation conventions and to take advantage of default variable types;
  (ii) clear identification of range, beginning and end of program loops;
  (iii) meaningful description of printed output;
  (iv) programming to achieve definite branching decisions in the computer program;
  (v) standards on the use of subscripted variables as subscripts;
  (vi) avoidance of mixed mode operations;
  (vii) standards for structuring arithmetic and logical expressions for clarity;
  (viii) standards for tests to detect invalid or indefinite intermediate results;
  (ix) standards on the use of nested constructs;
  (x) independence of module interfaces from changes to individual functions;
  (xi) recommendations on maximum module size;
  (xii) limitation of module parameters to the minimum number required;
  (xiii) standards for module entry and exit points;
  (xiv) standards on computer program readability;
  (xv) standards on efficient and reliable programming techniques.

Appendix E

Suggested Formats for SQA Forms

The following pages contain samples of the following Forms

1 Computer Code Classification
2 Computer Code Abstract
3 Change Request, Implementation and Control
4 Code Version Release
5 Analysis Definition, Implementation and Review

The layout of the forms is not important, however whatever layout is chosen should cover similar content to that shown.

Appendix E
Sample Formats for SQA Forms

Sample Form 1. Classification and Catalogue of Computer Codes

Classification and Description of Computer Codes managed by the xxx Unit				
Code	*Classification by Category & Type*	*Purpose and Use*	*Current Version & Date*	*Current Documentation*

Sample Form 2. Computer Code Abstract

Document U-yy-nn
Computer Code Abstract for the ... Program

Code Name	Version ID	Release Date	Code Owner
Description of Problem Solved			
Solution Methodology			
Hardware Platform(s)			
System Software Requirements			
Restrictions on use			
Authorised Users			
Code Manager & Contact Instructions			
User Manual		Theory Manual	

Appendix E: Sample Formats for SQA Forms

Sample Form 3. Change Request and Control Form (Page 1 of 2)

A: Change request - defined by Originator					
				Type of Request	
Program	Originator	Address	Date	Correct Nonconformance	Provide Enhancement
Describe Change Requested or Problem Encountered					
Routines Affected if Known					
Criteria Recommended for Acceptance of Change					
Path to Data Deck Used (if applicable)					
Other Comments					
Number of Additional Sheets attached					
B: Change Implementation & Verification Plan - Defined by Code Manager					
If Request not Accepted State Reason					
Request No	Date received	Assigned To	Complete By	Target Version	Issue Date
Routines Affected					
Theoretical and/or Numerical Basis					
Coding Instructions					
Benchmark Verification Cases					
Expected Effects					
Acceptance Criteria					
Updates to Manuals Required (if any)					
Work Assigned to	Date	Review Assigned to	Date	Copy to Originator	Date

Appendix E: Sample Formats for SQA Forms

Change Request and Control Form (Page 2 of 2)

C: Implementation in Development Version - Defined by Code Manager						
Coding and results reviewed by			**Comments**		**Date**	
Modification Accepted by code manager			**Date**		**Modification #**	
Integrated in Development Version			**Comments**		**Date**	
Verified & Accepted			**Comments**		**Date**	
Path to Modification Archive						
Documents Revised, date	**Abstract**		**User Manual**		**Theory Manual**	
Dates reported as Complete	**Developers:**		**Originator:**		**Users:**	

Sample Form 4. Notice of Release of New Production Version

Code Name	New Version ID	Effective Date	Previous Version ID	Withdrawal Date
Main Reason for Replacing Previous Version				
List of Modifications to Previous Version *(expand as required)*				
Mod ID	Date	Purpose & Effect		
Applicable Documentation, Revision Number and Date				
Abstract		User Manual		Theory Manual

Appendix E - Sample Formats for SQA Forms

Sample Form 5. Analysis Task Definition, Execution and Review

Part A - Planning to be completed by Project Leader

Analysis Project ID	Goal	Project Leader	Analysis Task ID	Goal	Assigned to
General statement of Project					
General statement of Task					
Task Details					
System to be Modelled					
Key Results sought					
Key Assumptions in modelling					
Source of specifications for Geometry					
Source of specifications for Operating Conditions					
Computer Code to be used			Version ID		
Code Documentation	User Manual			Theory Manual	
Relevant Validation reports					
Source of Standard Data Deck for this system					
Data Modifications required					
Special Instructions					
Target Completion dates	Data Deck			First Results	
	Final Analysis			Draft Report	
				Final Report	
Reviewer Assigned					

Part B - Review and Acceptance - to be completed as noted

Project ID		Task ID		Analyst		Reviewer	
Initials & Date	Completed by analyst	Reviewed by reviewer	Revised by analyst	Accepted by reviewer	Accepted by proj. ldr.
Data Deck					
First Results					N/A
Final Analysis					
Uncert. Anal.					
Draft Report					N/A
Final Report					
Archive of results & data	Path			Date	

Appendix F

Documentation Required by the Standards

Category	Category I Safety Critical		Category II Safety Related			Category III Noncritical	
Type	Type 1 Exists	Type 3 New	Type 1 Exists	Type 2 Modify	Type 3 New	Type1 Exists	Type 3 New
Standard	Q396.1.2	Q396.1.1	N286.7			Q396.2.2	Q396.2.1
Documentation Required							
Planning, Design & Development							
Project Quality Plan	X	X					X
Program Requirements Specification				x	X		
Contract Requirements Specification		X					X
Design & Development Plan		X		x	X		X
Program Design Description				x	X		
Documentation Control Plan	X	X					X
Verification Plan	X	X		x	X		X
Validation Plan	X	X		x	X		
Test Plan	X	X		x	X		
Verification Report				x	X		
Validation Report			X	x	X		
Test Procedures				x	X		
Test Report				x	X		
Configuration Mgt. Plan		X			X		X
Change Control Plan		X			X		X
Programming Manual					X		
Maintenance and Control							
Config. Mgt. Procedures	X	X	X	X	X	X	X
Change Control Procedures		X		X	X		X
Version History Record				X	X		
Release Control		X					X
Maintenance Manual					X		
SQA Review							
SQA manual	X	X				X	X
SQA procedures	X	X		X	X	X	X
SQA records	X	X		X	X	X	X
SQA audit	X	X				X	X
User Documentation							
Computer Program Abstract			X	X	X		
User's Manual			X	X	X		
Theory Manual			X	X	X		

Notes:

The standards require documentation of the topics marked X , but permit topics to be combined in a smaller number of reports as appropriate.

The topics marked x for type 2 programs are required to be documented for new developments only.