ORNL/TM-2016/80

Qualification of Simulation Software for Safety Assessment of Sodium-Cooled Fast Reactors: Requirements and Recommendations

Nicholas R. Brown
W. David Pointer
Matthew T. Sieger
George F. Flanagan
Wayne Moe, INL
Mark Holbrook, INL

April 2016

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2016/80

Reactor and Nuclear Systems Division

QUALIFICATION OF SIMULATION SOFTWARE FOR SAFETY ASSESSMENT OF SODIUM-COOLED FAST REACTORS: REQUIREMENTS AND RECOMMENDATIONS

Nicholas R. Brown
W. David Pointer
Matthew T. Sieger
George F. Flanagan
Wayne Moe, INL
Mark Holbrook, INL

Date Published: April 2016

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-BATTELLE, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES v
LIST OF TABLES v
ACRONYMS vii
ACKNOWLEDGEMENTS ix
1. INTRODUCTION 1
 1.1 BACKGROUND 1
 1.2 GOAL AND OBJECTIVES 1
 1.3 ASSUMPTIONS AND CONDITIONS 1
2. REQUIREMENTS AND GUIDANCE 2
 2.1 DOE-EH-4.2.1.2-Criteria 9
 2.2 EPRI Technical Report 3002002289 10
 2.3 NUREG/IA-0463 10
 2.4 NRC NUREG/BR-0167 11
 2.5 NUREG-1737 17
 2.6 DOE ORDER 414.D 17
 2.7 ASME NQA-1-2008/2009 STANDARD 18
 2.8 IEEE SOFTWARE DEVELOPMENT STANDARDS 19
  2.8.1 IEEE 1228-1994, Standard for Software Safety Plans [15] 19
  2.8.2 IEEE 1058-1998, Standard for Software Project Management Plans [16] 19
  2.8.3 IEEE 828-2005, Standard for Configuration Management is Systems and Software Engineering [17] 19
  2.8.4 IEEE 730-2002, Standard for Software QA Processes [18] 19
  2.8.5 IEEE 829-1998, Standard for Software and System Test Documentation [19] 20
  2.8.6 IEEE 1012-2004, Standard for System and Software V&V [20] 20
3. PRACTICES FROM OTHER CODES AND SOFTWARE PACKAGES 20
 3.1 SCALE 20
  3.1.1 Introduction to SCALE 20
  3.1.2 Summary of the SCALE QA Process 20
  3.1.3 Components of the SCALE QA Process 22
  3.1.4 SCALE QA Process Overview 23
  3.1.5 SCALE Testing, Validation, and Verification 23
  3.1.6 Lessons learned and Path Forward 26
 3.2 SAPHIRE 26
  3.2.1 Introduction to SAPHIRE 26
  3.2.2 Summary of SAPHIRE QA Process 26
  3.2.3 Components of the SAPHIRE QA Process 28
  3.2.4 Testing, V&V, and Internal, Automated, and External Testing 28
  3.2.5 V&V testing 30
  3.2.6 Lessons Learned 31
 3.3 RELAP-5-3D 32
  3.3.1 Introduction to RELAP5-3D 32
  3.3.2 Licensing Use of RELAP5-3D 32
  3.3.3 Summary of the RELAP5-3D QA Process 33
  3.3.4 Components of the RELAP5-3D QA Process 33
  3.3.5 RELAP-3D Testing, V&V 35
  3.3.6 Lessons Learned 36
 3.4 TRACE 37
  3.4.1 Introduction to TRACE 37

3.4.2 Summary of the TRACE QA Process 38
3.4.3 Components of the TRACE QA Process 38
3.4.4 TRACE Testing, Validation, and Verification 39
3.4.5 Lessons Learned and Path Forward 40
3.5 MPACT (CASL) 41
3.5.1 Introduction to MPACT 41
3.5.2 Summary of the MPACT QA Process 41
3.5.3 MPACT Testing, Validation and Verification 42
3.5.4 Lessons Learned and Path Forward 44
4. RECOMMENDATIONS FOR SFR SAFETY ANALYSIS CODE QUALIFICATION 44
4.1 Evaluation 45
4.2 NQA-1-2008/2009 compliant SQA program 46
4.3 SUPPORT for COMMERCIAL-GRADE DEDICATION 48
5. REFERENCES 49
APPENDIX A. OBJECT-ORIENTED AND PROCEDURAL TEST EXAMPLES A-1
APPENDIX B. CODE SNIPPET FROM MPACT UNIT TEST B-1
APPENDIX C. EXAMPLE CHECKLISTS FROM NUREG-1737 A C-1

LIST OF FIGURES

Fig. 1. Notional diagram of SCALE regression suite development process. 22
Fig. 2. CDash SCALE automated testing suite. 25
Fig. 3. SAPHIRE QA process. 27
Fig. 4. SAPHIRE release management process. 29
Fig. 5. Types of testing used during SAPHIRE development process. 29
Fig. 6. RELAP5-3D functional use analysis. 36
Fig. 7. NRC code development and assessment framework from Reference . 40
Fig. 8. Example regression test and solution from Reference 40. 44

LIST OF TABLES

Table 1. Cross index of QA activities with relevant requirements and guidance 5
Table 2. Summary of typical life cycle activities and documents [Table 1-1 in NUREG/BR-0167, 1993] 12
Table 3. Elements of software QA [Table 1 in NUREG-1737] 17
Table 4. Mapping of NUREG/BR-0167 guidelines to SCALE QA plan [23] 21
Table 5. Contacts and expectations for notification of safety-significant software errors [18] 24
Table 6. RELAP5-3D software QA tasks 34
Table 7. Fundamental validation assessment cases for TRACE, from Reference 34 39
Table 8. NRC regression suite for TRACE verification from Reference 34. 40
Table 9. Unit and regression test coverage in MPACT from Reference 36. 42
Table 10. Building blocks of core MPACT features covered by regression testing, adapted from Reference 36 42
Table 11. Plan for SQA Evaluation of existing safety analysis software (Reproduced from DOE-EH-4.3.1.2-Criteria Table 2-2) 45

ACRONYMS

ACRS	Advisory Committee on Reactor Safeguards
ASME	American Society of Mechanical Engineers
CAMP	Code Assessment and Maintenance Program
CASL	Consortium for Advanced Simulation of Light Water Reactors
CSAU	code scaling, applicability, and uncertainty
DOE	US Department of Energy
FCT	Fuel Cycle Technology
IEEE	Institute of Electrical and Electronics Engineers
INL	Idaho National Laboratory
IRUG	International RELAP5 User Group
LOCA	loss of coolant accidents
LWR	light water reactor
MPACT	Michigan Parallel Characteristics Transport Code
NEAMS	Nuclear Energy Advanced Modeling and Simulation
NPP	nuclear power plant
NRC	US Nuclear Regulatory Commission
PIRT	Phenomena Identification and Ranking Table
PRA	probabilistic risk assessment
QA	quality assurance
RCS	revision control system
RSICC	Radiation Safety Information Computational Center
SFR	sodium-cooled fast reactor
SP3	simplified P3 transport
SPC	Siemens Power Corporation3-D three-dimensional
SQA	software quality assurance
SQAP	software quality assurance program
V&V	verification and validation
VERA-CS	Virtual Environment for Reactor Applications Core Simulator

ACKNOWLEDGEMENTS

We gratefully acknowledge B. Rearden (ORNL), M. Muhlheim (ORNL), L. Briggs (ANL), A. Brunett (ANL), and T. Fanning (ANL) for their insightful review comments.

1. INTRODUCTION

1.1 BACKGROUND

As simulation software for nuclear system performance and safety assessment continues to evolve, so do the best practices, processes, and procedures for qualification of that software, especially for safety applications. Many guidance documents and industry standards offer recommendations, or even requirements, for successful qualification of code. They often reflect the unique qualities of the specific application for which they were first developed, and there is little consistency in the structure of one guidance document or standard versus another. The software qualifier is then left to piece together standards that satisfy the overarching guidance of the organization completing or reviewing the work.

In preparing this report, the authors reviewed not only relevant guidance documents and standards, but also the best practices of software packages and codes accepted as qualified by relevant regulatory bodies. From these reviews some common actions for code qualification can be identified:

1. In order to make an assessment, requirements must be clearly defined for both functional performance and application readiness.
2. All processes (e.g., project management, design, implementation, configuration management, validation, verification, noncompliance monitoring) must be documented and utilized.
3. A configuration management strategy must be established and utilized for all source code, test suites, test results, and documentation.
4. Source code must be verified, either manually or automatically, to confirm that models are implemented correctly and that source code is written correctly and operates correctly on all supported platforms.
5. Simulations of supported applications must be validated against experimental data.
6. Noncompliance and resulting actions must be documented.
7. The entire quality assurance (QA) program must be routinely audited to confirm compliance.

The following sections of this report provide additional details on guidance and requirements (Sect. 2), best practices adopted by qualified software teams as well as teams in the process of initially qualifying software (Sect. 3), and recommendations based on the authors' analysis of the experiences in qualification of software for regulatory use (Sect. 4).

1.2 GOAL AND OBJECTIVES

The goal of this review is to enable application of codes or software packages for safety assessment of advanced sodium-cooled fast reactor (SFR) designs. To address near-term programmatic needs, the authors have focused on two objectives. First, the authors have focused on identification of requirements for software QA that must be satisfied to enable the application of software to future safety analyses. Second, the authors have collected best practices applied by other code development teams to minimize cost and time of initial code qualification activities and to recommend a path to the stated goal.

1.3 ASSUMPTIONS AND CONDITIONS

In the development of this document, the authors have assumed that these requirements, guidelines, and best practices may be applied in support of qualifying one or more engineering software codes for the assessment of safety performance of advanced SFRs. The authors assume that the software will not be used as safety software providing real-time control of actuators or sensors in a nuclear facility. Furthermore, the authors assume that, while the software may be used to support safety-significant decision making, the software should never be used as the sole basis for decision making. The authors anticipate that the users of the qualified code are reactor designers, license applicants, evaluators of completed assessments, and regulators. The authors anticipate that the users of this report and recommendations contained herein are primarily associated with the code development team rather than end users. The authors further anticipate that this document will used to support baseline code qualification efforts rather than qualification of a specific safety or performance analysis, and do not intend for this report or recommendations contained herein to be used as a surrogate for relevant regulations and guidance.

To provide more focused comments, the authors have assumed that the initial application of these recommendations will focus on the SFR system safety performance code SAS4A/SASSYS. However, the report is expected to provide useful guidance for any nuclear code developers interested in using their code in performing safety analyses that may be used in the regulatory process regardless of reactor technology option.

2. REQUIREMENTS AND GUIDANCE

For U.S. nuclear power plants and fuel reprocessing plants, the QA requirements are set by 10 CFR Part 50 Appendix B. The U.S. Nuclear Regulatory Commission (NRC) oversees the licensing and operation of such plants. The licensing process ensures that the requirements of 10 CFR Part 50 Appendix B are met.

The NRC Regulatory Guide 1.28 *Quality Assurance Program Criteria (Design and Construction)* [1] describes the methods that the NRC considers acceptable for complying with 10 CFR 50 Appendix B. The regulatory position stated therein explicitly endorses the requirements included in the American Society of Mechanical Engineers (ASME) NQA-1-2008 and the NQA-1a-2009 Addenda (collectively referred to as NQA-1-2008/2009) [2] as providing an acceptable basis for complying with the CFR, subject to a set of additions and modifications (concerned with recordkeeping and audits) specified in Section C of the Guide.

Specific requirements applicable to software are defined in NQA-1-2008/2009 Part II Subpart 2.7 *Quality Assurance Requirements for Computer Software for Nuclear Facility Applications*. Guidance for the implementation of Part II Subpart 2.7 requirements is given in Part IV Subpart 4.1 *Application Appendix: Guide on Quality Assurance Requirements for Computer Software*.

The NQA-1-2008/2009 standard requires that software not produced under a QA program compliant with the standard be dedicated in accordance with the requirements of Part II, Subpart 2.14, *Quality Assurance Requirements for Commercial Grade Items and Services*. The Electric Power Research Institute (EPRI) Technical Report 3002002289, *Plant Engineering: Guidelines for the Acceptance of Commercial-Grade Design and Analysis Computer Programs Used in Nuclear Safety-Related Applications* (2013) [3] provides detailed guidance for licensees and nuclear suppliers regarding the dedication process. It is this process that licensees are likely to attempt to apply to SAS4A/SASSYS, and so care should be taken to fully understand the supplier QA processes and evidence needed to support commercial-grade dedication.

For existing data used in software validation activities, the guidance in NQA-1-2008/2009 Part III Subpart 3.3, *Nonmandatory Appendix 3.1: Guidance on Qualification of Existing Data* should be used.

NUREG/IA-0463 (2014) presents a report issued by the International Regulator Task Force on Safety Critical Software [4]. This report identifies common (consensus) positions on the technical basis for licensing reviews of safety-critical software for nuclear reactors. While the report is not a product of the US NRC or other US government agency, the NRC published it as a NUREG/IA document because it considers the content a valuable technical reference that can be used for future improvements in NRC's regulatory guidance and to inform current licensing reviews. The report provides excellent guidance on the specific criteria that new and existing software applications are evaluated against for licensing approval. The focus is on digital control systems for safety purposes, but guidance is given as well for a graded approach applicable to existing software used in safety-related applications in Section 1.11 of the document.

NUREG/BR-0167, *Software Quality Assurance Program and Guidelines* [5] was first published in 1993 to provide guidance to NRC organizations and contractors in the development and maintenance of software for use by the NRC staff. The report defines the expected quality-assured development process, identifies required documentation, and, in some cases, provides example documentation outlines. NUREG/BR-0167 is not explicitly based upon the requirements of NQA-1-2008/2009 Part II Subpart 2.7, and so care should be taken to ensure that all requirements are fulfilled if the guidance is used. The NRC provides supporting guidance and checklists specific to thermal hydraulic codes in NUREG-1737, *Software Quality Assurance Procedures for NRC Thermal Hydraulic Codes* [6].

For DOE activities with the potential to affect nuclear safety, the source of QA requirements is 10 CFR 830 Subpart A; the NRC does not license DOE facilities. The regulatory regime is therefore different.

QA requirements for activities regulated by the U.S. Department of Energy (DOE) are given in DOE Order 414.1D [7]. DOE encourages contractors to apply appropriate industry standards in the development of QA plans and documentation that are consistent with the quality rigor level associated with anticipated end use, and explicitly endorses the use of NQA-1-2008/2009 for activities related to nuclear safety, including software. The aforementioned requirements stated in NQA-1-2008/2009 Part II Subpart 2.7 are therefore regarded as an acceptable means of complying with DOE O 414.1D and 10 CFR 830 Subpart A for software development.

DOE describes suggested approaches to satisfy SQA requirements for nuclear safety-related applications in DOE Guide 414.1-4, *Safety Software Guide for Use with 10 CFR 830 Subpart A, Quality Assurance Requirements and DOE O 414.1C, Quality Assurance* [8], which was published in 2005.

An especially relevant and useful reference is DOE-EH-4.2.1.2-Criteria *Software Quality Assurance Plan and Criteria for the Safety Analysis Toolbox Codes* [9]. This report outlined the plan to qualify six computer codes used primarily for accident analysis; ALOHA, CFAST, EPIcode, GENII, MACCS2, and MELCOR, many of which had an uncertain SQA pedigree. Although the plan focuses on qualification of the codes for DOE use and not specifically for NRC licensing, many of the activities defined in this document are directly relevant to the problem of upgrading SAS4A/SASSYS SQA for licensing use. Of particular interest are the requirements and procedures described in Tables 3-3.1 through 3-3.8.

The DOE Office of Nuclear Energy’s Fuel Cycle Technologies (FCT) Program has established a QA Program Document [10], revised in 2012, which provides further guidance on assessing quality rigor level, identification of related requirements, and implementation of appropriate QA activities as part of product development within that program. The DOE Office of Nuclear Energy’s Nuclear Energy Advanced Modeling and Simulation (NEAMS) Program has developed similar guidance in a Software QA Plan [11], published in 2013.

In the course of satisfying SQA requirements, the software developer may apply one or several industry standards to support the specific implementation strategy adopted. Table 1 shows how relevant orders, standards and guidance relate to software quality assurance activities. The table provides only an index to support further review by the developer and does not provide a comprehensive mapping of activities.

Relevant industry standards may include

- ASME verification and validation (V&V) 20, Standard for V&V in Computational Fluid Dynamics and Heat Transfer.
- ASME V&V 30, Standard for V&V of Software for Nuclear Applications
- ANSI/ANS-10.4-1987 (R1998), Guidelines for the V&V of Scientific and Engineering Computer Programs for the Nuclear Industry, American Nuclear Society, 1987.
- ANSI/ANS-10.7-2013: Non-Real-Time, High-Integrity Software for the Nuclear Industry--Developer Requirements, American Nuclear Society, 2013.
- IEEE standard series:
  - IEEE Std 610.12-1990 (R2002), IEEE Standard Glossary of Software Engineering Terminology, The Institute of Electrical and Electronics Engineers, 1990, E-ISBN 0-7381-0391-8.

- IEEE Std 730-2002, IEEE Standard for Software Quality Assurance Plans, The Institute of Electrical and Electronics Engineers, 2002, ISBN 0-7381-3258-3.
- IEEE Std 828-2005, IEEE Standard for Software Configuration Management Plans, The Institute of Electrical and Electronics Engineers, 2005, ISBN 0-7381-4764-8.
- IEEE Std 829-1998, IEEE Standard for Software Test Documentation, The Institute of Electrical and Electronics Engineers, 1998, ISBN 0- 7381-1443-X.
- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications, The Institute of Electrical and Electronics Engineers, 1998, ISBN 0-7381-0332-2.
- IEEE Std 1008-1987 (R2002), IEEE Standard for Software Unit Testing, The Institute of Electrical and Electronics Engineers, 1986, E-ISBN 0-7381-0400-0.
- IEEE Std 1012-2004, IEEE Standard for Software V&V, The Institute of Electrical and Electronics Engineers, 2005, ISBN 0-7381-4641-2.
- IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions, The Institute of Electrical and Electronics Engineers, 1998, ISBN 0-7381-1455-3.
- IEEE Std 1028-1997, IEEE Standard for Software Reviews, The Institute of Electrical and Electronics Engineers, 1998, ISBN 1- 55937-987-1.
- IEEE Std 1044-1993 (R2002), IEEE Standard Classification for Software Anomalies, The Institute of Electrical and Electronics Engineers, 1994, ISBN 0-7381-0406-X.
- IEEE Std 1058-1998, IEEE Standards for Software Project Management Plans, The Institute of Electrical and Electronics Engineers, 1998, ISBN 0-7381-1447-2.
- IEEE Std 1061-1998 (R2004), IEEE Standard for Software Quality Metrics Methodology, The Institute of Electrical and Electronics Engineers, 1998, ISBN 0-7381-1059-6.
- IEEE Std 1074-2006, IEEE Standard for Developing Software Life Cycle Processes, The Institute of Electrical and Electronics Engineers, 2006, ISBN 0-7381-4956-X.
- IEEE Std 1228-1994, IEEE Standard for Software Safety Plans, The Institute of Electrical and Electronics Engineers, 1994, ISBN 0-7381- 0420-5.
- IEEE Std 1233-1998, IEEE Guide for Developing System Requirements Specifications, The Institute of Electrical and Electronics Engineers, 1998, ISBN 0-7381-0337-3.

The IEEE standards are not explicitly focused on nuclear code qualification, but they provide useful guidance for code QA in general.

It should be noted that it is the licensee who is ultimately responsible for meeting the requirements listed above. The goal of the software owners should be to support that licensing process with sufficient evaluation, verification, validation, configuration control, and problem reporting processes to enable the software’s use in the licensing arena.

Guides and standards that were identified as particularly helpful are further discussed in sections 2.1-2.x below.

Table 1. Cross index of QA activities with relevant requirements and guidance

Software QA Activity	NQA-1-2008/2009 Part II Subpart 2.7	NUREG/IA-0463	NUREG/BR-0167	NUREG-1737	DOE-EH-4.2.1.2-Criteria	DOE G 414.1-4	Other useful standards
Software Project Management	Part I, Requirement 1 Organization	Section 1.6 Organizational Requirements	Section 5 Project Management, Section 4.2 Software Project Plan	Section 3.1.1		Activity 1 – Software project management and quality planning	IEEE 1058
Quality Assurance Program & Plan	Part I, Requirement 2 QA Program	Section 1.7 Software Quality Assurance Program and Plan	Section 8 Quality Assessment and Improvement		Table 3-3.1.2 SQA Procedure/Plans,		ANSI/ISO/ASQ Q9001-2015 IEEE 730 IEEE 1028 IEEE 1061 IEEE 1228
Software Project Risk Management			Section 5.8 Risk Management			Activity 2 – Software Risk Management	
Configuration Management	Section 203 Software Configuration Management	Section 2.7 Change Control and Configuration Management	Section 6 Configuration Management	Section 5.0 Configuration Control	Table 3-3.7.12 Configuration Control	Activity 3 – Software configuration management	IEEE 828 IEEE 1008 IEEE 1233
Procurement & Supplier Management	Section 300 Software Acquisition	Section 1.4 Pre-existing Software (PSW)			Table 3-3.1.3 Dedication	Activity 4 – Procurement and Supplier Management	
Requirements	Section 401 Software Design Requirements	Section 2.1 Computer Based System Requirements, Section 2.3 Software Requirements, Architecture and Design	Section 2.2 Requirements Definition	Section 3.2 Requirements Definition, Section 4.3 Software Requirements Documentation	Table 3-3.1.5 Requirements Phase	Activity 5 – Software requirements identification and management	IEEE 830
Design	Section 402 Software Design	Section 2.2 Computer System Architecture and Design, Section 2.3 Software Requirements, Architecture and Design	Section 2.3 Design, Section 4.4 Design Documentation	Section 3.3 Software Design, Section 4.4 Design Documentation	Table 3-3.3.6 Design Phase	Activity 6 – Software design and implementation	IEEE 1016
Implementation	Section 403 Implementation	Section 2.4 Software Implementation	Section 2.4 Implementation, Section 4.5 Implementation Documentation	Section 3.4 Coding	Table 3-3.4.6 Design Phase, Table 3-3.4.7 Implementation Phase	Activity 6 – Software design and implementation	

Software Failure Analysis		**Section 1.2 System Classes, Function Categories and Graded Requirements for Software,** **Section 1.13 Software Reliability**					
Support Software	Section 600 Support Software	Section 1.5 Tools					
Testing	Section 404 Acceptance Testing	Section 2.5 Verification	Section 2.5 Qualifications Testing	Section 3.5 Validation Testing	Table 3-3.5.8 Testing Phase	Activity 8 – Verification and Validation (V&V)	ANSI/ANS-10.4 IEEE 829 IEEE 1012
Reviews	Section 202 Review	Section 1.11 Graded Requirements for Safety Related Systems (New and Pre-existing Software)			Table 3-3.1.4 Evaluation		
Installation & Acceptance	Section 404 Acceptance Testing	Section 2.6 Validation and Commissioning	Section 2.6 Installation and Acceptance	Section 3.6 Installation and Acceptance, Appendix C	Table 3-3.6.10 Acceptance Test	Activity 3 – Software Configuration Management	ANSI/ANS-10.4 IEEE 828 IEEE 829 IEEE 1233
Operation & Maintenance	Section 405 Operation, Section 406 Maintenance	Section 2.8 Operational Requirements	Section 2.7 Operations and Sustaining Engineering	Section 4.0 Error Corrections and Code Maintenance	Table 3-3.7.11 Operation and Maintenance	Activity 3 – Software Configuration Management, Activity 9 – Problem reporting and corrective action	IEEE 828
Verification & Validation	Section 402.1 Software Design Verification, Section 404 Acceptance Testing	Section 2.5 Verification, Section 2.6 Validation and Commissioning	Section 3 V&V, Section 4.6 V&V Documentation	Sections 3.2.2, 3.3.2, 3.4.2, 3.5, 3.6.4, 3.6.5		Activity 8 – V&V	ANSI/ANS-10.4 ASME V&V 20 ASME V&V30 IEEE 1012 NUREG/CR-5249
Problem Reporting & Corrective Action	Section 204 Problem Reporting and Corrective Action		Section 7 Nonconformance Reporting and Corrective Action	Section 4.0 Error Corrections and Code Maintenance	Table 3-3.8.13 Error Impact	Activity 9 – Problem reporting and corrective action	IEEE 730 IEEE 828 IEEE 1044

Training			**Section 5.7 Training**			**Activity 10 – Training of Personnel in the Design, Development, Use, and Evaluation of Safety Software**	
Documentation	Section 201 Documentation		Section 4 Documentation and Deliverables	Sections 3.1.1, 3.1.2, 3.2.1, 3.5.1, 3.6.3, 3.6.5, 5.1, 6.0, Appendix A, Appendix B	Table 3-3.6.1 User Instructions Table 3-3.1 through 3-3.8 give specific documentation requirements	Activities 1-10	
Audit & Assessment	Part I, Requirement 18 Audits	Section 1.10 Independent Assessment	Section 8 Quality Assessment and Improvement				
Retirement	Section 407 - Retirement		Section 2.8 Retirement and Archiving				

2.1 DOE-EH-4.2.1.2-CRITERIA

DOE-EH-4.2.1.2-CRITERIA was published by the Defense Nuclear Facilities Safety Board (DNFSB) in 2002 to provide a plan, criteria for the qualification, and implementation procedures to upgrade the SQA for six safety-related computer codes used primarily for accident analysis [9]. To fulfill the requirements of 10 CFR 830, the Board chose ASME NQA-1-2000 Part II Subpart 2.7 as a baseline standard, and used the requirements therein to set criteria for evaluating the codes. The evaluation plan (Table 2-2 in the report) covered the major requirements of NQA-1-2000 with a procedural basis for evaluating software that was developed mostly outside of NQA-1 requirements, and used in accident analysis applications. The procedural basis provided instructions for evaluation of existing accident analysis software, referencing detailed criteria based on NQA-1-2000 compliant requirements (listed in Table 3-3 of the document).

The plan identified the necessary documentation to be produced:

- Software Quality Assurance Plan
- Software Requirements Document
- Software Design Document
- Test Case Description and Report
- Software Configuration and Control Document
- Error Notification and Corrective Action Report
- User's Manual, and other relevant documentation (model description, weekly or monthly reports to code sponsor, etc.).

The overall process of SQA evaluation was summarized in eight steps (Section 4.0, Table 4-1):

1. Establish SQA criteria for the codes (Section 3 of the document summarizes the results of this activity)
2. Provide requested documentation, and estimate resources needed to address deficiencies (Appendix E summarizes the results of a similar activity for the MACCS2 code)
3. Review the documentation and evaluate the software against the SQA criteria
4. Document code review in "gap" analysis reports
5. Determine minimum required actions to be taken before software will meet SQA criteria
6. SQA documentation upgrade and modification of software to address deficiencies or make improvements
7. Identify upgraded computer code version to DOE users to central registry (establish baseline version)
8. Provide configuration management and control of qualified software.

The detailed criteria, plans, and implementation procedures given in DOE-EH-4.2.1.2-CRITERIA were developed for a situation closely matching that of SAS4A/SASSYS, and provide excellent guidance. However, some key differences are relevant and should be taken into consideration before adopting their approach wholesale:

- The process targets DOE requirements and not specifically those of the NRC.
- The version of NQA-1 used is older than the currently endorsed version (NQA-1-2000 vs. NQA-1-2008/2009), and differences in the standards must be carefully evaluated and applied.
- The commercial-grade dedication process defined in NQA-1-2008/2009 Part II, Subpart 2.14, *Quality Assurance Requirements for Commercial Grade Items and Services* was significantly revised from NQA-1-2000, is likely to be applied by NRC licensees, and should be used as an additional source of criteria.

2.2 EPRI TECHNICAL REPORT 3002002289

In order to be accepted for use in nuclear safety-related applications, software not designed and manufactured in accordance with a quality assurance program that meets the requirements of 10 CFR Part 50, Appendix B must be dedicated for use in accordance with the requirements of 10 CFR, Part 21, which sets requirements for the use of "commercial-grade" items in applications important for nuclear safety. EPRI Technical Report 3002002289 [3] was published in 2013 to provide guidance to the nuclear industry for the commercial-grade dedication of design & analysis computer programs used in nuclear safety-related applications. Section 6 of the EPRI report describes a template process by which this dedication may be accomplished, and provides several examples.

The commercial-grade dedication process in no way absolves the supplier of the need to perform careful SQA and documentation of the software. A key critical characteristic for dedication is the degree of "built-in" quality as assured by a structured development process, documentation, conformance to standards, internal reviews and verifications, thoroughness of testing, training and qualification of developers, and effective oversight. The critical characteristics described in Tables 6-2 through 6-6 of the report provide a set of criteria that should be viewed as a potential source of acceptance criteria for the SAS4A/SASSYS SQA upgrade effort (keeping in mind that the specific set of critical characteristics will vary from licensee to licensee and from application to application).

2.3 NUREG/IA-0463

NUREG/IA-0463 is a result of a technical exchange between the Regulator Task Force on Safety Critical Software (TF SCS) for nuclear reactors and the NRC seeking consistency in the technical basis for the licensing review of software in safety-related digital instrumentation and control systems for nuclear power plants, and was published by the NRC in December 2015 [4]. The TF SCS is a group of experts from regulatory and safety authorities in Belgium, Canada, Finland, Germany, Spain, Sweden, and the United Kingdom as well as their technical-support organizations. The major result of the work is the identification of consensus and common technical positions (requirements) on a set of important licensing issues raised by the design and operation of computer based systems used in nuclear power plants for the implementation of safety functions.

The task force adopted the view that three basic independent types of evidence can and must be produced: evidence related to the quality of the development process; evidence related to the adequacy of the product; and evidence of the competence and qualifications of the staff involved in all of the system life cycle phases. In addition, convincing operating experience may be needed to support the safety demonstration of pre-existing software.

The issue areas were partitioned into two sets: "generic licensing issues" and "life cycle phase licensing issues". Issues in the second set are related to a specific stage of the computer based system design and development process, while those of the former have more general implications and apply to several stages or to the whole system lifecycle:

PART 1: GENERIC LICENSING ISSUES
- 1.1 Safety Demonstration
- 1.2 System Classes, Function Categories and Graded Requirements for Software
- 1.3 Reference Standards
- 1.4 Pre-existing Software (PSW)
- 1.5 Tools
- 1.6 Organizational Requirements
- 1.7 Software Quality Assurance Program and Plan

1.8 Security
1.9 Formal Methods
1.10 Independent Assessment
1.11 Graded Requirements for Safety Related Systems (New and Pre-existing Software)
1.12 Software Design Diversity
1.13 Software Reliability
1.14 Use of Operating Experience
1.15 Smart Sensors and Actuators

PART 2: LIFE CYCLE PHASE LICENSING ISSUES
2.1 Computer Based System Requirements
2.2 Computer System Architecture and Design
2.3 Software Requirements, Architecture and Design
2.4 Software Implementation
2.5 Verification
2.6 Validation and Commissioning
2.7 Change Control and Configuration Management
2.8 Operational Requirements

Each section discusses the rationale for the issue, lists specific concerns, and presents consensus requirements and recommended practices. Many of the requirements and recommendations are applicable to digital control systems fulfilling safety functions; however, guidance is given on applying a graded application of requirements to pre-existing software performing a safety-related function, which is a category more applicable to SAS4A/SASSYS.

2.4 NRC NUREG/BR-0167

NUREG/BR-0167 was first published in 1993 to provide guidance to NRC organizations and contractors in the development and maintenance of software for use by the NRC staff. NUREG/BR-0167 [5] provides specific requirements for QA and assessment in all phases of the software development life cycle. The document defines six steps in the software development life cycle: Requirements Definition, Design, Implementation, Qualification Testing, Installation and Acceptance, and Operations and Sustaining Engineering. Expectations are then outlined for planning, execution, testing, and documentation in each of these areas. This matrixed approach to software QA is summarized in Table 2.

This guidance on software development and qualification in NUREG/BR-0167 [5] should not be confused with the guidance in NUREG/CR-5249, *Quantifying Reactor Safety Margins* [12], which is focused on qualification of analyses of a particular safety-relevant event. The latter document defines the code scaling, applicability, and uncertainty (CSAU) evaluation methodology. The CSAU methodology is specific to a single analysis of a particular event or series of events occurring in a particular system design.

Table 2. Summary of typical life cycle activities and documents [Table 1-1 in NUREG/BR-0167, 1993]

	Requirements definition	**Design**	**Implementation**	**Qualification testing**	**Installation and acceptance**	**Operations and sustaining engineering**
Principal technical activities performed	-Analyze requirements	-Develop preliminary design -Develop detailed design -Develop preliminary user documentation	-Develop unit designs and unit code	-Conduct qualification testing in accordance with the qualification test plan and qualification test procedures	-Install the software on the target computer -Conduct acceptance testing in accordance with the acceptance test plan and procedure	-Perform all of the activities of development, as appropriate. -Perform sustaining engineering activities to ensure that the original capabilities and design remain intact.
V&V activities performed	-Conduct requirements inspection -Conduct software requirements review	-Conduct design inspections Plan qualification and acceptance test -Conduct preliminary design review -Conduct critical design review	-Develop unit and integration test plans and procedures -Conduct unit and integration testing -Develop qualification and acceptance test procedure	-Witness qualification tests	-Witness acceptance tests	-Perform all V&V activities as appropriate
Documentation and deliverables developed	-Software requirements documentation -Overall V&V plan -Software project plan	-Software design documentation -Qualification test plan -Acceptance test plan -Preliminary user's documentation	-Qualification test procedures -Acceptance test procedure -Unit and integration test results	-Qualification test report -Nonconformance reports based on test results -Final user's documentation	-Acceptance test report -Nonconformance reports based on test results	-Update all documents, as required -Develop new documentation, as appropriate

Table 2. Summary of typical life cycle activities and documents [Table 1-1 in NUREG/BR-0167, 1993] (continued)

	Requirements definition	**Design**	**Implementation**	**Qualification testing**	**Installation and acceptance**	**Operations and sustaining engineering**
Project management activities performed	-Develop software project plan -Ensure users participate in requirements definition -Conduct tracking and oversight activities	-Conduct tracking and oversight activities -Re-plan as required	-Conduct tracking and oversight activities -Re-plan as required	-Conduct tracking and oversight activities -Re-plan as required	-Conduct tracking and oversight activities -Re-plan as required	-Conduct tracking and oversight activities -Re-plan as required
Configuration management activities performed	-Develop or update configuration management procedures -Place software requirements under configuration control (i.e., establish the requirements baseline)	-Place software design documentation under configuration control	Place code and qualification test documentation under configuration control	-Place software design documentation under configuration control (i.e., establish the product baseline)	-Place software design documentation under configuration control (i.e., establish the product baseline)	-Maintain the baseline and developmental configuration -Establish new product baseline and operational baseline
Nonconformance reporting and corrective action activities performed	-Develop or update nonconformance reporting and corrective action procedures -Document requirements documentation nonconformance	-Document design and requirements documentation nonconformance	-Document design and requirements documentation nonconformance	-Document code and qualification test non-conformance -Document design and requirements documentation nonconformance	Document code acceptance test documentation -Document design and requirements documentation non-conformance	-Document all nonconformances as applicable

Table 2. Summary of typical life cycle activities and documents [Table 1-1 in NUREG/BR-0167, 1993] (continued)

Requirements definition	Requirements definition	Design	Implementation	Qualification testing	Installation and acceptance	Operations and sustaining engineering
Quality assessment and improvement activities performed	-Assess software requirements documentation and software project plan -Assess requirement definition process -Initiate product and process improvement activities, as required	-Assess software design -Assess the qualification test plan and acceptance test plan -Initiate product and process improvement activities, as required	-Assess unit design, unit code, unit test plans, integration test plans, and integration test procedures -Assess implementation process -Assess qualification test procedures and acceptance test procedures	-Assess qualification test results -Assess qualification test process -Initiate product and process improvement activities, as required	-Assess qualification test results -Assess qualification test process -Initiate product and process improvement activities, as required	-Assess all products and processes -Initiate product and process improvement activities, as required

2.5 NUREG-1737

The NRC published NUREG-1737 [6] in 2000 to document procedures used by the NRC and its contractors in the qualification of thermal hydraulic codes for use by the NRC. NUREG-1737 [6] references the basic requirements defined in NUREG/BR-0167 [5], but it takes the additional step of defining best practices in satisfying those requirements for each step in the software product development life cycle. NUREG-1737 [6] describes characteristics of acceptable procedures, outcomes, and documentation and includes appendices that provide recommendations for preparing the software QA plan and checklists to ensure that major components of each step have been completed.

The major activities described in NUREG-1737 [6] and relevant checklists are outlined in Table 3.

Table 3. Elements of software QA [Table 1 in NUREG-1737]

Life cycle	Development product	Verification & validation activities	Checklist
Initial Planning	SOW, Project Plan SQA Plan	Management Review	QA Forms 03-06
Requirements Definition	Software Requirements Specifications	Verification of Requirements Review of test plan and acceptance criteria	QA Form 04
Software Design	Software Design and Implementation Document	Review of Design	QA Forms 07-08
Coding	Source Code Verification Testing Report	Review/Inspection of Source Code Verification of Program Integration Verification of Test Results	QA Form 05
Software Testing	Validation Testing Report	Validation of Program	QA Forms 05-09
Installation and Acceptance	Installation Package	Verification of Installation Package	QA Forms 12, 10, and 11
	Upgrading Program Documentation	Verification of Program Documentation	

2.6 DOE ORDER 414.D

The DOE published Order 414.D [7] in 2011 to provide broad QA guidance to ensure that all products from DOE and from the National Nuclear Security Administration meet customers' expectations and requirements. The order primarily focuses on assignment of responsibilities but also clearly requires that all DOE programs implement a graded QA program and that they define a procedure for QA program approval and changes and procedures for evaluation of technical capability and qualifications. The order references several general quality program consensus standards, including

- ASME NQA-1-2008 with the NQA-1a-2009 addenda, QA Requirements for Nuclear Facility Applications;
- ANSI/ISO/ASQ Q9001-2008, Quality Management System-Requirements;
- ANSI/ASQ Z 1.13-1999, Quality Guidelines for Research;
- DOE-STD-1150-2002, QA Functional Area Qualification Standard; and
- DOE STD-1172-2003, Safety Software QA Functional Area Qualification Standard.

Order 414.1D [7] itself is not specific to software QA. However, DOE Guide 414.1-4 was published in 2005 as a supplementary, nonmandatory guidance document to clarify application of Order 414.1C [8] to software used in DOE safety analyses. Order 414.1C [8] was superseded by Order 414.1D [7] when it was issued in 2011. DOE Guide 414.1-4 defines 10 activities that met the requirements of Order 414.1C [8]:

- Activity 1: Software Project Management and Quality Planning
- Activity 2: Software Risk Management
- Activity 3: Software Configuration Management
- Activity 4: Procurement and Supplier Management
- Activity 5: Software Requirements Identification and Management
- Activity 6: Software Design and Implementation
- Activity 7: Software Safety
- Activity 8: V&V
- Activity 9: Problem Reporting and Corrective Action
- Activity 10: Training Personnel in the Design, Development, Use, and Evaluation of Safety Software

Updated guidance has not been provided to address change resulting from implementation of Order 414.D.

A general QA plan was developed for the FCT Program in 2012. A QA plan, which includes specific software QA guidance, was published for the Consortium for Advanced Simulation of Light Water Reactors (CASL) in 2012 as CASL-U-2012-0047, *Quality Manual* [13], and revised in 2016 as CASL-U-2016-1070, *CASL-QA-001 CASL Quality Assurance Program Plan, Revision 4.0* and CASL-U-2015-0010 *CASL-QA-030 CASL Software Quality Assurance Requirements*, both designed to meet the criteria of NQA-1-2008/2009 Part II Subpart 2.7, under a graded approach defined under the guidance of NQA-1-2008/2008 Part IV Subpart 4.2, *Guidance on Graded Application of the Nuclear Quality Assurance (NQA) Standard for Research and Development* [8]. A software QA plan was published for the NEAMS Program in 2013 as LLNL-SM-455533 [11]. None of the above examples directly addresses the requirements and concerns regarding NRC licensing.

2.7 ASME NQA-1-2008/2009 STANDARD

The NRC and DOE explicitly endorse ASME NQA-1-2008 and the NQA-1a-2009 Addenda (collectively referred to as NQA-1-2008/2009) [14] as providing an acceptable basis for complying with the requirements of 10 CFR 50 Appendix B and 10 CFR 830 Subpart A, respectively. Specific requirements applicable to software are defined in NQA-1-2008/2009 Part II Subpart 2.7 *Quality Assurance Requirements for Computer Software for Nuclear Facility Applications*. Guidance for the implementation of Part II Subpart 2.7 requirements is given in Part IV Subpart 4.1 *Application Appendix: Guide on Quality Assurance Requirements for Computer Software*.

The NQA-1-2008/2009 standard requires that software not produced under a QA program compliant with the standard be dedicated in accordance with the requirements of Part II, Subpart 2.14, Quality Assurance Requirements for Commercial Grade Items and Services.

Existing data used in software validation activities should use the guidance in NQA-1-2008/2009 Part III Subpart 3.3, Non-mandatory Appendix 3.1: Guidance on Qualification of Existing Data.

In summary, NQA-1-2008/2009 Part II Subpart 2.7 and related Part I requirements, primarily Requirements 3 (Design Control/Section 800 Software Design Control) and 11 (Test Control/Section 400 Computer Program Test Procedures), are recommended as the primary set of SQA criteria for the evaluation of safety-related computer software. This selection is based on

- Nuclear industry precedent with ASME NQA standards
- Federal and commercial sectors continued involvement with, and maintenance of the ASME NQA standards

- Quality assurance perspective through connection with 10 CFR 50 Appendix B, 10 CFR 830 Subpart A and 10 CFR 70
- Independence of roles in developing and maintaining software, among management, work performers, and work reviewers
- Graded application based on safety, risk, and hazard analysis of the function of the software
- Focus on protection of the public and workers
- Long-standing presence and incorporation with many DOE contractors' quality assurance programs, with focus on nuclear safety, and
- Completeness and relevance to scientific, applied research, design, analysis and nuclear engineering software.

2.8 IEEE SOFTWARE DEVELOPMENT STANDARDS

The Institute of Electrical and Electronics Engineers (IEEE) develops and maintains a significant ecosystem of technical standards to support software development and software QA for a wide variety of end-use applications. Many of these standards are referenced in the above requirements and guidance documents. A small selection of IEEE standards is highlighted in this section because they provide specific recommendations that may provide a basis for the development of documentation that satisfies the above requirements.

2.8.1 IEEE 1228-1994, Standard for Software Safety Plans [15]

The Software Safety Plans standard provides minimal acceptable requirements for "the Plan used for the development, procurement, maintenance, and retirement of safety-critical software." This plan defines the overall approach to software development and maintenance and is supported by a number of other documents, including the five additional plans described in this section.

2.8.2 IEEE 1058-1998, Standard for Software Project Management Plans [16]

The Software Project Management Plan is the controlling document for managing a software project, which defines all of the processes, both technical and managerial that will be applied in the project. The standard is developed with a broad view of activities constituting a "software project" and may be equally applicable to projects that develop new source code or that focus on modifications to existing source code. This document encompasses the scope of the project, the work plan, control plans, risk management plans, and the project closeout plan.

2.8.3 IEEE 828-2005, Standard for Configuration Management is Systems and Software Engineering [17]

The Software Configuration Management Plan standard is primarily focused on definition of the configuration management process in order to directly support implementation of configuration management controls. However, the standard does include (in Annex D) an outline of the Configuration Management Plan document and specific requirements for each section.

2.8.4 IEEE 730-2002, Standard for Software QA Processes [18]

The IEEE Standard for Software QA Processes "establishes requirements for initiating, planning, controlling, and executing the Software QA (SQA) processes of a software development or maintenance project." The standard is primarily focused on the development of the SQA process itself. Annex C outlines the process for developing an SQA plan and provides a mapping between sections of an SQA plan outline and the sections of the standard. These tables provide a reference for expected SQA plan content.

2.8.5 IEEE 829-1998, Standard for Software and System Test Documentation [19]

The IEEE Standard for Software and System Test Documentation defines the minimum requirements for test processes and tasks in six areas: Management, Acquisition, Supply, Development, Operation, and Maintenance Process. Requirements are defined for documentation of processes and reporting of outcomes of each testing application.

2.8.6 IEEE 1012-2004, Standard for System and Software V&V [20]

The IEEE Standard for System and Software V&V defines those processes that "determine whether the development products of a given activity conform to the requirements of that activity and whether the product satisfies its intended use and user needs." The standard covers the full extent of V&V activities required within a software project, not just verification of the source code and validation of the models implemented within the code. Section 12 of the standard provides an outline of the Software V&V Plan and defines requirements for each major section.

3. PRACTICES FROM OTHER CODES AND SOFTWARE PACKAGES

The following subsections are intended to inform on key questions, including:

1. What are key software packages doing to comply with NRC QA requirements?
2. What are the opportunities to learn from these experiences to develop and implement effective software QA processes?

Note that this section is intended as a series of examples, and identifies practices used in a variety of code packages with different approaches to meeting NRC QA requirements.

3.1 SCALE

3.1.1 Introduction to SCALE

This subsection is focused on conveying the highlights of SCALE QA implementations. The SCALE computer software system, developed at Oak Ridge National Laboratory, has a variety of light water reactor analysis and fuel cycle applications (e.g., nuclear criticality safety, lattice physics, depletion analysis, cross section sensitivity). In addition, SCALE integrates generalized capabilities for Monte Carlo particle transport and radiation shielding. The first distribution of SCALE occurred in 1980 on behalf of the NRC, who is the original and a sustaining sponsor of SCALE. SCALE now has a global distribution to approximately 7,000 users in 56 nations.

3.1.2 Summary of the SCALE QA Process

The first QA program for SCALE was initiated in 1989. A key component of the QA program is the corresponding SCALE QA plan. The QA plan originated with the initiation of the overall QA program, and has been updated several times. The most recent revision occurred in 2013 [21] and constituted a major enhancement of the SCALE QA process. Previous SCALE QA programs focused on configuration management, but the latest QA program features an expanded scope [22]. The scope of the present QA program includes continuous integration, as well development planning and coordination. The objective of the present SCALE QA program is to [22]:

1. ensure new features are designed to meet needs of users,
2. ensure new features perform as designed,
3. establish quality control to ensure existing features continue to perform as designed,
4. establish configuration control for traceability,
5. coordinate efforts for consistency and efficiency, and
6. identify defects and provide corrective actions.

The QA policies of the SCALE project stipulate that [22]

1. all procedures are to be understood and followed,
2. users are promptly informed of issues,
3. safety-significant issues are reported to users and sponsors on a timely basis,
4. all staff members strive to deliver defect-free software with validated performance, and
5. all staff members subscribe to the Continuous Integration model of software development, where the production code base is maintained in a shippable, defect-free state at all times.

The present iteration of the SCALE QA plan was developed independently of the software-specific QA requirements of the NRC, as documented in NUREG/BR-0167 [5]. However, the general (not software-specific) QA requirements documented in 10 CFR 830 Subpart A [8] were considered [8]. Although NUREG/BR-0167 [5] was written in 1993, the SCALE QA plan was not mapped to the requirements in NUREG/BR-0167 [5] until after it was revised in 2013. However, the SCALE QA plan meets or exceeds all of the requirements outlined by the NRC in NUREG/BR-0167 [5]. The relationship between the sections of the SCALE QA plan and the corresponding NRC QA requirements is shown in Table 4. The SCALE QA plan is also compliant with ISO Q9001 (2008 edition) [22] and DOE Order 414.1D.

Table 4. Mapping of NUREG/BR-0167 guidelines to SCALE QA plan [23]

NUREG BR-0167 requirement/guidance	Section in the SCALE QA plan or SCALE procedure
2.2 Requirements Definition	6.2, 6.4, SCALE-CMP-001, SCALE-CMP-013
2.3 Design	6.2, 6.4, SCALE-CMP-001, SCALE-CMP-013
2.4 Implementation	6.4.6, SCALE-CMP-001, SCALE-CMP-013
2.5 Qualification Testing	6.4.5, 6.6.2, SCALE-CMP-013
2.6 Installation and Acceptance	N/A
2.7 Operations and Sustaining Engineering	All sections of the SCALE QA plan, SCALE-CMP-001, SCALE-CMP-013, SCALE-CMP-004, SCALE-CMP-012
2.8 Retirement and Archiving	N/A
3.2 V&V Activities	6.2, 6.4, 6.6, 4.6, 7.2, SCALE-CMP-001, SCALE-CMP-012, SCALE-CMP-013
4 Documentation and Deliverables	3.4
5 Project Management	All sections of SCALE QA Plan, SCALE-CMP-001, SCALE-CMP-013, SCALE-CMP-004, SCALE-CMP-012
6 Configuration Management	3.4, 5.2, 6.4.6, 6.6, 6.6.3, 7.3, SCALE-CMP-001, SCALE-CMP-013
7 Nonconformance Reporting and Corrective Action	7.4, 7.6.2, SCALE-CMP-004
8 Quality Assessment and Improvement	7,7.3.2, 3.4.4, ORNL SBMS Procedure "Audits and Assessments"

The specific quality objectives of the SCALE project QA plan include [21]

1. registration to the ANSI/ISO/ASQ Q9001-2008 standard,
2. responses to user inquiries are provided within 5 days of receipt,

3. responses to user discrepancy reports are provided within 48 hours of receipt, and
4. safety-significant issues are reported to users and sponsors within 48 h of that determination.

These objectives enable a rapid, systematic, and effective response from the perspective of an end user.

3.1.3 Components of the SCALE QA Process

Early SCALE QA plans focused primarily on configuration management, where changes to the source code were documented to clearly quantify when a feature was introduced or retired. Insufficient focus on comprehensive testing often led to the discovery of discrepancies after a release. These discrepancies often required a patch to be issued for the software or data in question, causing end users to repeat code qualification certifications, and reducing confidence in the tools.

The foundation of the modern SCALE QA process was improved with the release of SCALE 6.1. During the summer of 2010, a team of students mentored by ORNL staff completed a comprehensive acceptability testing of SCALE tools [22]. The fundamental approach was to examine the functional needs of each tool. Then each tool, treated as a "black box," was applied in a series of rigorous and realistic end-use cases. Over 200 previously unknown defects were identified spanning modules, functionalities, and end-use applications. This test matrix was captured as a "regression suite." The fundamental approach developed during this effort has progressed to include data testing, with one example being the thousands of unique tests for neutron and gamma transmissions through all materials at many energies.

Additional improvements in SCALE QA included the unification of all source code (eliminating differences in module source code for various platforms) and an automated testing system to enable application of the regression suite [22].

Several important lessons were learned from the improved SCALE QA process. One approach that was demonstrated was to identify the present state of quality in a particular tool. The approach in 2010 was to "seal off" the tool, consider its functional needs, and create a regression suite based on those needs. The regression suite, developed using engineering judgment, informs on the progression of the output of the tool for a given perturbation. A simplified notional flow diagram of the approach is shown in Fig. 1. Another key lesson is that a graded approach is recommended, bearing in mind the relative "importance" of a particular tool or particular results.

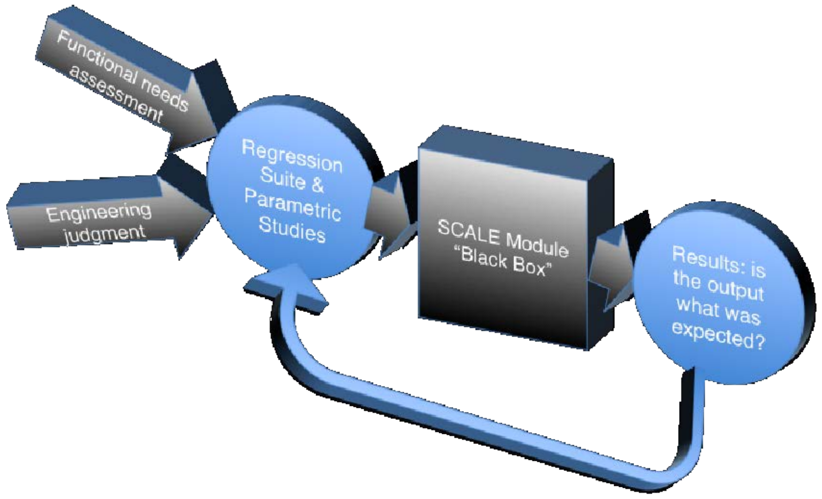

Functional needs assessment
Engineering judgment
Regression Suite & Parametric Studies
SCALE Module "Black Box"
Results: is the output what was expected?

Fig. 1. Notional diagram of SCALE regression suite development process.

Once existing SCALE features were well quantified and sufficient tests were available, a continuous-integration system was created to automatically run the test suite with each update to the code repository. Any changes in previously accepted results are captured and the developers are notified to begin corrective action. The availability of a more comprehensive test suite and continuous-integration system are directly related to a substantial reduction in the defect rate of SCALE for version 6.1 in 2011 relative to version 6.0 in 2009. Through two years of deployment, SCALE 6.0 required 10 patches to be issued, with the first patches appearing just weeks after initial release. With more robust QA, SCALE 6.1 has only required two patches, one of which was driven by an error in the nuclear data provided to all codes from the National Nuclear Data Center and the other was driven by a code error for a realistic, but rarely used, calculation performed by an external user.

With the increase quality, the user base of SCALE has greatly expanded from approximately 2,500 users in 2009 to approximately 7,000 users in 2016. This decrease in need to continually review and correct previously deployed features has provided new opportunities for the development team to focus on modernizing the legacy code base and develop many innovative advanced capabilities.

3.1.4 SCALE QA Process Overview

The foundation of the SCALE QA process is the *SCALE Procedure for Feature Changes* [24]. As stated in the procedure:

> A SCALE Quality Assurance Feature Case in the electronic tracking system must be completed any time a change is made to SCALE, whether it is a correction or an enhancement. The Kanban process is used to track the progress of each Feature through design, implementation, documentation, testing, and deployment. The Kanban steps are: Proposed, Approved, In Progress, In Testing, Ready to Ship, and Deployed.

The electronic tracking system provides a record of compliance with QA practices throughout the entire software life cycle. Changes to the software repository to implement a feature, test cases, and test results are integrated within the tracking system for seamless traceability.

A vital component of the SCALE QA plan is the procedure for discrepancy reports [22]. A discrepancy report is used to "(1) identify a situation where the SCALE code system fails to perform according to the documentation, (2) indicate the impact to current and past users, and (3) recommend action to resolve and/or temporarily circumvent the discrepancy" [22]. The SCALE procedure for discrepancy reporting is outlined in detail in Reference 21. In particular, discrepancies that are identified to be significant software errors are those that "occur with no warning or error messages, appear to allow proper execution of the software yet provide results that are: (1) Inconsistent with the evaluated nuclear data or the theory models applied in the codes, and (2) Judged to be of potential significance to operational safety (e.g., potential k_{eff} error greater than 1%)."

One of the core drivers for the most recent SCALE QA plan was a quality incident that occurred in 2005, known as the KENO "hole" error [22]. This error was significant enough that it resulted in "stop work" orders at both the High Flux Isotope Reactor at ORNL and the Spent Fuel Storage Facility at Idaho National Laboratory [22]. The error, which was due to numerical round off in particularly sensitive geometries, resulted in the issuance of an NRC information notice [25]. This particular error was considered to be a safety-significant software error [22]. These errors trigger specific responses as documented in Reference 22 (see Table 5).

3.1.5 SCALE Testing, Validation, and Verification

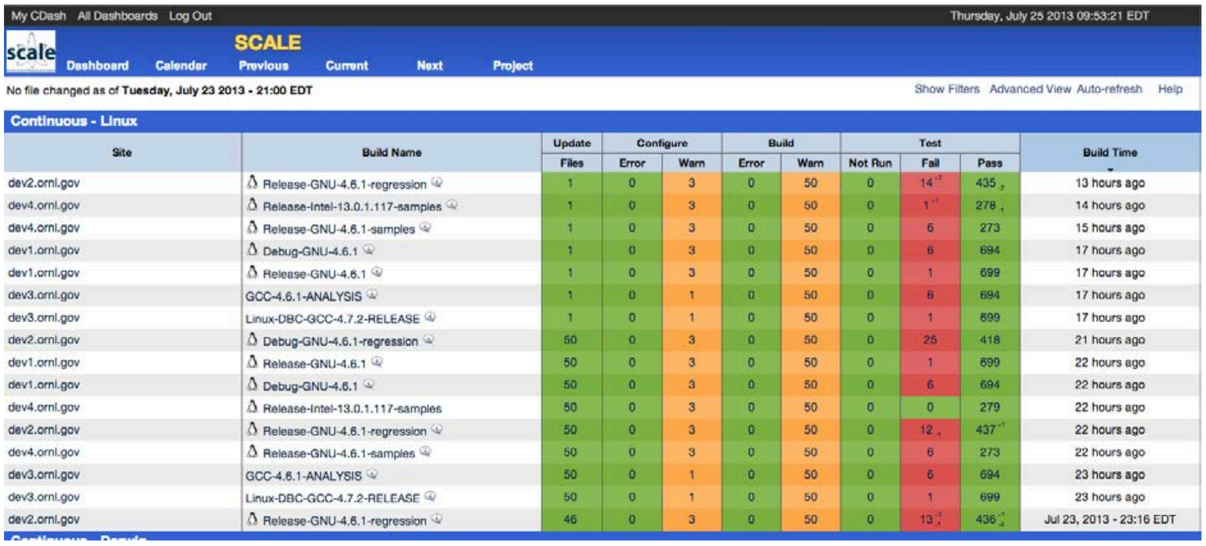
A foundation of any modern software project is the ability to routinely compile and test the software and data and provide continual support for the latest hardware and compilers using a continuous-integration system. After each incremental update to the SCALE source code repository the continuous-integration system automatically builds the code on dozens of systems, including Linux, Mac, and Windows with different compilers and compiler options. On each system, a suite of over 2000 test cases is run to quantify the performance of the update and its impact on any other features. This rigorous testing is performed dozens of times each day, resulting in the quantification of performance with approximately 150,000 tests per day. The results of the tests and the associated changes are reported to an internal website, known as the SCALE Dashboard. All developers can review the Dashboard to monitor the performance of numerous SCALE features on different platforms with different compilers using a pass/fail metric without the need to configure and run all of these tests themselves.

Table 5. Contacts and expectations for notification of safety-significant software errors [22]

Organization/group	Points of contact/expectations
DOE	• Nuclear Criticality Safety Program Manager • Packaging Certification Program Manager • Nuclear Fuels Storage and Transportation Planning Project National Technical Director Interacts with SCALE Project Leader to understand error, judge impact on operational safety, review checklist, and make decision on issuing as *Significant Software Error*. Coordinates notification issuance for any errors deemed *significant* and interacts with DOE offices, DOE facilities, and other government organizations.
NRC	• Office of Nuclear Material Safety and Safeguards SCALE Project Manager • Office of Nuclear Regulatory Research SCALE Project Manager Interacts with SCALE Project Leader to understand error, judge impact on operational safety, review checklist, and make decision on issuing as *Significant Software Error*. Coordinates notification issuance for any errors deemed *significant* and interacts with NRC offices, licensees, and other government organizations.
Radiation Safety Information Computational Center (RSICC)	Provides notice in RSICC Newsletter and via e-mail alert to recipients of the code version(s) affected.
Code Developers	SCALE Project Leader – Prepares and reviews checklist. Interacts with DOE, NRC, and RSICC. Issues the notification to any user groups pertinent to the software. Notification of errors (or discrepancies where code does not perform as described by documentation) that are not deemed to be *significant* may use this checklist format as deemed appropriate. Notifies those on the *SCALE News* email list (scalenews@home.ornl.gov) and assures the information is posted in the SCALE website.

SCALE testing is an automated procedure using three types of tests [26]. Unit tests are targeted tests aimed at isolating specific functionalities, regression tests are integral tests designed to run quickly, and sample problems are intended to demonstrate a key feature of the code to end users. SCALE consists primarily of object-oriented code (e.g., C++ or Java) or procedural code (e.g., Fortran 90, Fortran 2003). For the object-oriented code, approximately 61% of classes in SCALE are covered by unit tests (as a specific example, ORIGEN as a module is above average, with roughly 70% coverage). Ideally, for a code written with an object-oriented programming paradigm, every class will have a unit test. Some tests will be very simple, others more complex. An example of the Dashboard interface is shown in Fig. 2, summarizing the passed and failed tests on various builds of pre-release development features.

For a code written in a language that is not object oriented, the objective of one unit test per class is not relevant. However, subroutines and specific functionalities are relevant, so a unit test per subroutine or specific functionality is one potential objective. Included in this report are examples of SCALE unit tests for C++ (object-oriented) code and Fortran 90 (procedural) code. Because these example tests are not within the SCALE code, they are not export controlled.



No file changed as of **Tuesday, July 23 2013 - 21:00 EDT**

Continuous - Linux

Site	Build Name	Update	Configure		Build		Test			Build Time
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	
dev2.ornl.gov	Release-GNU-4.6.1-regression	1	0	3	0	50	0	14	435	13 hours ago
dev4.ornl.gov	Release-Intel-13.0.1.117-samples	1	0	3	0	50	0	1	278	14 hours ago
dev4.ornl.gov	Release-GNU-4.6.1-samples	1	0	3	0	50	0	6	273	15 hours ago
dev1.ornl.gov	Debug-GNU-4.6.1	1	0	3	0	50	0	6	694	17 hours ago
dev1.ornl.gov	Release-GNU-4.6.1	1	0	3	0	50	0	1	699	17 hours ago
dev3.ornl.gov	GCC-4.6.1-ANALYSIS	1	0	1	0	50	0	6	694	17 hours ago
dev3.ornl.gov	Linux-DBC-GCC-4.7.2-RELEASE	1	0	1	0	50	0	1	699	17 hours ago
dev2.ornl.gov	Debug-GNU-4.6.1-regression	50	0	3	0	50	0	25	418	21 hours ago
dev1.ornl.gov	Release-GNU-4.6.1	50	0	3	0	50	0	1	699	22 hours ago
dev1.ornl.gov	Debug-GNU-4.6.1	50	0	3	0	50	0	6	694	22 hours ago
dev4.ornl.gov	Release-Intel-13.0.1.117-samples	50	0	3	0	50	0	0	279	22 hours ago
dev2.ornl.gov	Release-GNU-4.6.1-regression	50	0	3	0	50	0	12	437	22 hours ago
dev4.ornl.gov	Release-GNU-4.6.1-samples	50	0	3	0	50	0	6	273	22 hours ago
dev3.ornl.gov	GCC-4.6.1-ANALYSIS	50	0	1	0	50	0	6	694	23 hours ago
dev3.ornl.gov	Linux-DBC-GCC-4.7.2-RELEASE	50	0	1	0	50	0	1	699	23 hours ago
dev2.ornl.gov	Release-GNU-4.6.1-regression	46	0	3	0	50	0	13	436	Jul 23, 2013 - 23:16 EDT

Fig. 2. CDash SCALE automated testing suite.

Two example unit tests for SCALE are included in this report, one for an object-oriented programming paradigm, the other for a procedural programming paradigm (see Appendix A).

The automated unit test included for the object-oriented programming paradigm is for ORIGEN and tests concentration unit conversion in SCALE. This example unit test is intended to verify that concentration units are accurately converted by the ORIGEN code.

The automated unit test included for the procedural programming paradigm is for ORIGEN and tests concentration unit conversion in SCALE. This example unit test is intended to verify that nuclide identifiers are interpreted correctly. This example is presented to show a typical procedural unit test in SCALE.

Validation of SCALE calculations is based on many series of established experiments, including:

- approximately 400 criticality and shielding benchmarks,
- over 100 isotopic assays measurements for spent nuclear fuel,
- over 100 decay heat measurements from spent fuel assemblies,
- gamma spectra measurements from fission burst experiments, and
- neutron spectra measurements from spent fuel and (α ,n) sources.

Although these test suites are not part of the continuous-integration system due to their long runtimes, the test suites are run by analysts after each major enhancement to code capabilities of nuclear data libraries.

In depth verification is performed primarily using code-to-code comparison with other production-level tools that provide similar, yet independent features. Thousands of test cases are applied ranging from very simple systems to confirm physics and nuclear data, even for a single nuclide interaction at a single energy, to extremely complex scenarios such as the depletion of a reactor core and characterization of isotopic inventories. Through this robust verification, issues in SCALE can be identified and corrected, and issues in the other tools are often identified and communicated to those teams as well.

3.1.6 Lessons learned and Path Forward

The first step in designing a regression suite is a functional needs assessment to determine what the code should do and what its range of applicability is. One good rule of thumb is that if any functionality is discussed in the manual, there should be unit tests associated with the key subroutines for that functionality.

A key source for integral/regression tests in SCALE is user feedback from training/sample problems as well as beta testing. This is where the extensive SCALE training and user base helps enable enhanced QA. A key recommendation based on the SCALE experience is that efforts are needed to expand the user base for SAS4A/SASSYS, where possible, and most importantly to build and open community among existing users for information exchange. In SCALE, many regression tests are accumulated over time based on user bug reports. Additionally, if there is a new feature or change, a regression test is automatically added for SCALE.

3.2 SAPHIRE

3.2.1 Introduction to SAPHIRE

The NRC has sponsored development of a personal computer software application for use in performing probabilistic risk assessments (PRAs). The tool is called Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE). In nuclear power plant applications, SAPHIRE is used to model plant responses (and associated uncertainties) to initiating events, quantify associated core damage frequencies, and identify important contributors to core damage (Level 1 PRA). It can also evaluate elements such as containment failure and release models for severe accident conditions given the occurrence of core damage (Level 2 PRA). SAPHIRE can be employed for limited risk quantifications associated with radiological release consequences (Level 3 PRA).

3.2.2 Summary of SAPHIRE QA Process

The SAPHIRE QA manual identifies key methodologies used to systematically plan and maintain software quality [21]. These activities are identified in Fig. 3 and work together to ensure high levels of quality throughout the code development cycle.

Currently, SAPHIRE (Version 8) follows guidance established in NUREG/BR-0167 [5], “Software Quality Assurance Program and Guidelines” [22]. These requirements are supplemented by comparison against best software engineering methods as provided in IEEE Std. 1012-2004 [20], “IEEE Standard for Software V&V” (which is a secondary requirements resource). A detailed checklist was created to map key requirements derived from these documents against appropriate software-related development parameters and provide a “Pass/Fail” grading system during subsequent evaluations [27].

Specific types of documentation that supports the released version of SAPHIRE include:

- design documents
- Software Project Plan
- System Test Plan
- Acceptance Test Plan
- Quality Assurance Plan

- Configuration Management Plan
- Software V&V Plan (including a requirements traceability matrix)

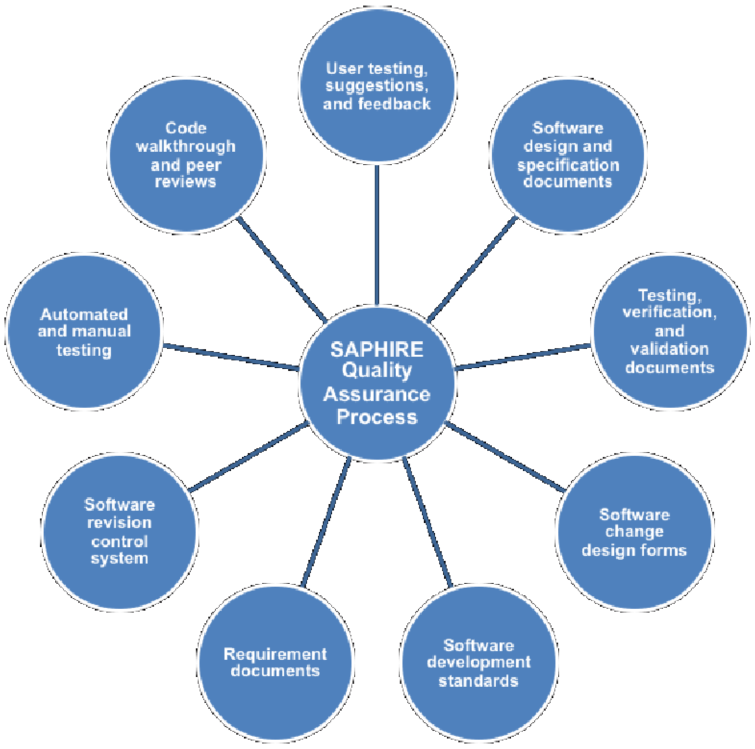

SAPHIRE Quality Assurance Process
User testing, suggestions, and feedback
Software design and specification documents
Testing, verification, and validation documents
Software change design forms
Software development standards
Requirement documents
Software revision control system
Automated and manual testing
Code walkthrough and peer reviews

Fig. 3. SAPHIRE QA process.

Per NUREG/BR-0167 guidance, SAPHIRE is classified as “Level 1,” which corresponds to technical application software used in safety decisions. SAPHIRE is also assigned an integrity level of “1” under IEEE Standard 1012-2004 software integrity level requirements, which is the lowest level on a scale corresponding to the likelihood of occurrence of an operating state that contributes an error and an error consequence.

The current version of IEEE Std. 1012-2004 defines integrity level as “a value representing project-unique characteristics (e.g., complexity, criticality, risk, safety level, security level, desired performance, and reliability) that define the importance of the system, software, or hardware to the user.” Integrity levels are used to determine the V&V tasks, activities, rigor, and the level of intensity of the V&V to be performed. As noted in the standard: “The degree of rigor and intensity in performing and documenting the task shall be commensurate with the integrity level. As the integrity level decreases, so does the required scope, intensity, and degree of rigor associated with the V&V task. For example, a hazard analysis performed for integrity level 4 software might be formally documented and consider failures at the module level; a hazard analysis for integrity level 3 software may consider only significant software failures and be documented informally as part of the design review process.”

Additionally, the IEEE standard provides a table that establishes “error consequence.” The standard notes that: “Each cell in the table assigns an integrity level based on the combination of an error consequence and the likelihood of occurrence of an operating state that contributes to the error. Some table cells reflect more than one integrity level, indicating that the final assignment of the integrity level can be selected to address the system application and risk mitigation recommendations. For some industry applications, the definition of likelihood of occurrence categories may be expressed as probability figures derived by analysis or from system requirements."

3.2.3 Components of the SAPHIRE QA Process

SAPHIRE software QA (SQA) requirements are contract driven and interpreted from DOE Order 414.1C, "Quality Assurance," 10 CFR 830 Subpart A, "Nuclear Safety Management," and ASME NQA-1-2000, "Quality Assurance Requirements for Nuclear Facility Applications" [8]. SAPHIRE must also conform to all internal Idaho National Laboratory (INL) program documents that address software development and configuration management.

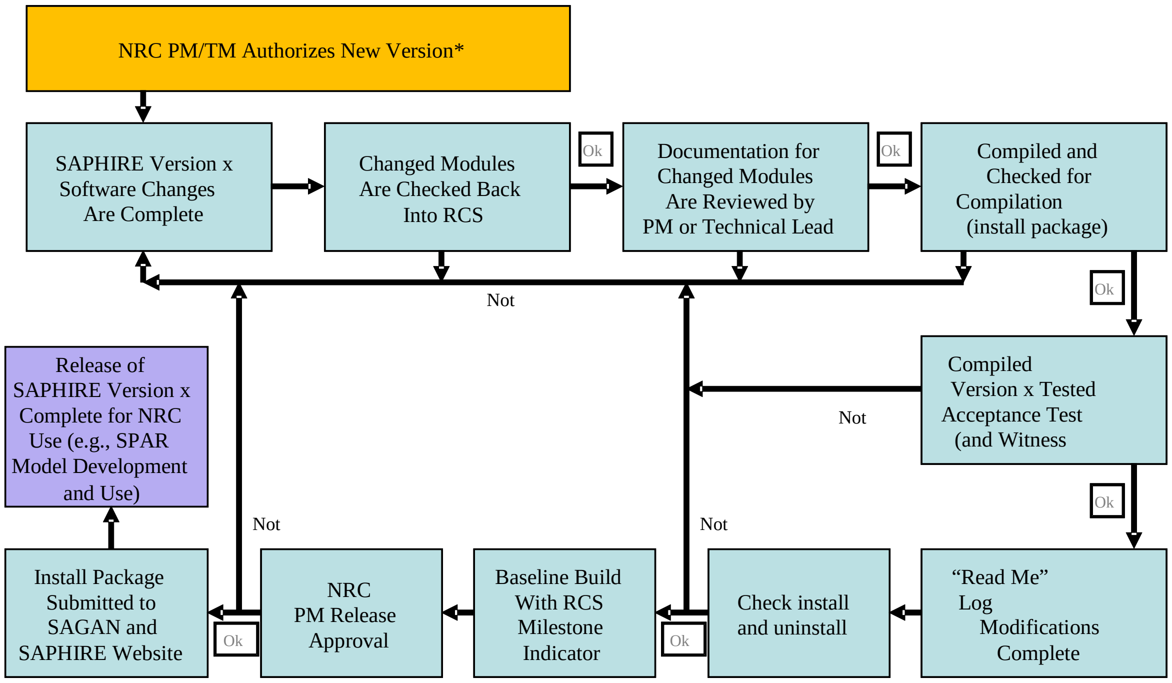
A number of specific controls consistent with the QA program elements identified in Fig. 3 are imposed by contract to ensure that only authorized changes are made in response to user-reported bugs, suggestions, and enhancements. In addition, the configuration management approach includes software release verification checklists and requirements for developers to perform and pass a suite of tests prior to new version release [28].

Because SAPHIRE is sponsored directly by the NRC and is routinely used by regulators and licensees in high-consequence nuclear safety decisions, a sophisticated version release management process was established between code developers and the NRC. This process focuses on up-front inputs and approvals directly from NRC staff, includes software quality assurance elements, and is illustrated in Fig. 4.

The four boxes on the top-right corner and down the right side of Fig. 4 (i.e., starting with the box "Documentation for Changed Modules are Reviewed by…") represent the software QA elements of the SAPHIRE release management process.

3.2.4 Testing, V&V, and Internal, Automated, and External Testing

SAPHIRE development utilizes a multifaceted testing approach. The acceptance testing program is described in NUREG/CR-7039, "Systems Analysis Programs for Hands-on Integration Reliability Evaluations (SAPHIRE) Version 8–Volume 6 Quality Assurance," [29] and the software configuration management plan (INL/EXT-09-16696) [28]. The testing approach (Fig. 5) is composed of three items: internal testing, automated testing, and external testing.


NRC PM/TM Authorizes New Version*
SAPHIRE Version x Software Changes Are Complete
Changed Modules Are Checked Back Into RCS
Ok
Documentation for Changed Modules Are Reviewed by PM or Technical Lead
Ok
Compiled and Checked for Compilation (install package)
Not
Ok
Release of SAPHIRE Version x Complete for NRC Use (e.g., SPAR Model Development and Use)
Compiled Version x Tested Acceptance Test (and Witness
Not
Ok
Not
Not
Install Package Submitted to SAGAN and SAPHIRE Website
Ok
NRC PM Release Approval
Baseline Build With RCS Milestone Indicator
Ok
Check install and uninstall
"Read Me" Log Modifications Complete

Completed By:

Date: Version #:

*Release decision predicated on:
(1) Milestones in NRC requirements documents , or
(2) INL/NRC evaluate and NRC approves when a new should be released based on severity (critical) or quantity of fixes and improvements.

Fig. 4. SAPHIRE release management process.

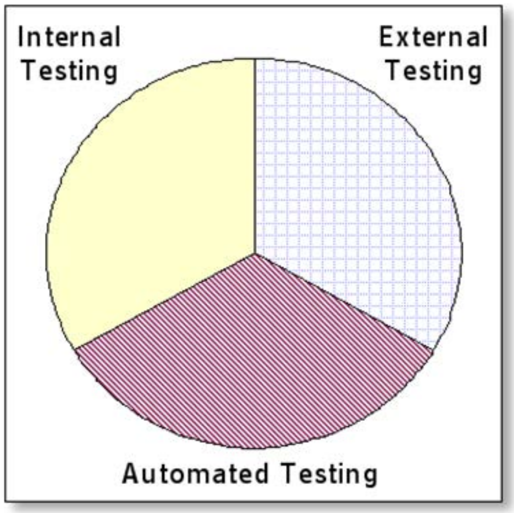


Fig. 5. Types of testing used during SAPHIRE development process.

Internal testing (or developmental testing) includes checks performed by the development team itself to ensure quality during the development process. The test suite is evaluated against significant changes and new features. New tests are developed to check a new feature when the developer and customer agree that it is appropriate. To develop a new test, a suitable test scenario with a database and validated correct answers must be determined.

Prior to official release, the software is run through a series of automated regression acceptance tests. These automated regression tests exercise the code to identify inconsistencies in results. The focus is on scripts that simulate user input rather than actual interface testing. One hundred and five tests simulate user input to the SAPHIRE through a test script and results are captured and compared to expected results. This approach ensures that given a static input PRA file, the risk or reliability results from SAPHIRE are consistent from one release to the next. The test scripts mimic actions taken by an analyst and might include starting SAPHIRE and navigating the user interface by selecting menu options, clicking buttons, and typing information. A summary and a detailed report of the results of the tests are produced so that an overview of the results can quickly be determined and any failures (or successes) can be traced in more detail.

External testing are evaluations performed by risk and reliability end users using, in many cases, “real world” models. This would typically include NRC review and beta testing before the software version is released using the approval path shown in Fig. 4. Beta testing helps ensure results produced by the new version are correct and that the software is user-friendly and functional. Beta testers are analysts experienced with PRA methods and terminology and typically are familiar with earlier versions of SAPHIRE.

All code source files are maintained in a controlled library server with processes in place to uniquely identify all components, modules, documentation, error reports, test suites, and test results.

3.2.5 V&V testing

Independent software V&V testing is a formal process for ensuring that the software development process results in a high-quality software product. For SAPHIRE, independent V&V testing is performed in accordance with INL/EXT-09-15649, *SAPHIRE 8 Software Independent Verification and Validation Plan* [30].

Independent V&V of a software product should be done by an organization that is both technically and managerially separate from the organization responsible for developing the product. The reasons for performing software V&V includes:

- Verification that determines whether the product meets its requirements.
- Validation that determines if the product performs its intended activities

Recent independent V&V evaluation results concerning SAPHIRE were documented in the Version 8 Final Report [31]. The independent V&V team reviewed code documentation and examined patterns of action needed to provide confidence that the software product conformed to technical requirements established in NUREG/BR-0167 [5], and IEEE Std. 1012-2004 [20]. The findings from this report cited the following best practices:

1. The software development process is improved by conducting independent V&V activities.
2. SAPHIRE development implemented and followed requirements for Level 1 software as defined in Section 1.2 of NUREG/BR-0167 [5].
3. The NRC (primary code user) is expected to perform an audit of software QA implementation once a year against the requirements of NUREG/BR-0167.
4. SAPHIRE configuration management for version control, source code, and documentation uses a Revision Control System (RCS).

5. The organization, tasks, roles, and responsibilities of the SAPHIRE team are defined.
6. The peer reviews and code walkthroughs are being implemented.
7. The INL SAPHIRE team has a defined quality assessment and improvement approach.

3.2.6 Lessons Learned

Additionally, the independent V&V team noted a number of lessons learned [31]. These include:

1. Ensure all software requirements are documented. These requirements form the basis of software plans, products, and activities. Ensure that documented requirements define the needed software response to anticipated classes of input data (including erroneous data) and provide information and details necessary to design the software (e.g., mathematical models, equations, and data requirements). Because requirements inevitably change as a project evolves, manage requirements throughout the development and maintenance effort in accordance with well-defined change control procedures.
2. Ensure all software requirements are uniquely defined as functional, performance, design constraints, attributes, and external interface requirements. With respect to the application of these terms, IEEE Std. 830-1993, "Recommended Practice for Software Requirements Specifications", states that: 1) Functionality addresses what the software is supposed to do, 2) External Interfaces address software interaction with people, the system's hardware, other hardware and other software, 3) Performance addresses the speed, availability, response time, recovery time of various software functions, 4) Attributes are concerned with the portability, correctness, maintainability, security, and similar considerations, 5) Design constraints are imposed on implementation and are concerned with required standards in effect, implementation language, policies for database integrity, resource limits, operating environments, etc.
3. Ensure all software requirements are testable. If a software requirement is not testable then it should not be considered a functioning software requirement.
4. Conduct a software requirements review at the end of requirements definition to ensure the intent, completeness, verifiability, consistency, and technical feasibility of the requirements.
5. Ensure that all software design components are documented and meet the requirements defined in software requirements documentation.
6. Ensure all software design components are uniquely defined and specify the overall software structure so that they can be translated into code.
7. Conduct a preliminary design review when the preliminary design (software architecture) has been established to ensure that the preliminary design is complete (meets all the requirements), verifiable (through testing or other means), consistent, and technically feasible.
8. Conduct a critical design review when design is complete to ensure that the design is indeed complete (meets all the requirements and meets design completion criteria), verifiable (through testing or other means), consistent, and technically feasible.
9. Ensure the requirements traceability matrix shows all software requirements as mapped to design components and test cases.
10. Conduct formal peer inspections to find errors.
11. Ensure that documentation is maintained.

It is important to remember that a software component is an identifiable part of a larger program or construction. Usually a component provides a particular function or group of related functions. In programming design, a system is divided into components that in turn are made up of modules. "Component test" means testing all related modules that form a component as a group to make sure they work together. In object-oriented programming, a component is a reusable program building block that can be combined with other components in the same. Examples of a component include a single button in a graphical user interface, a small calculator, and an interface to a database manager.

Design components in the SAPHIRE context formulate the basic building blocks of SAPHIRE. There is a cut set generation component, a basic event editor component, a fault tree editor component, an event tree editor component as well as others. Each component is documented and meets requirements outlined in the applicable requirements document. Since the completion of the independent V&V assessment, the practice has been to document significant changes in external (i.e., not part of the code) design documents.

3.3 RELAP-5-3D

3.3.1 Introduction to RELAP5-3D

The RELAP5-3D code is an outgrowth of the one-dimensional RELAP5/MOD3 code developed at INL for the NRC. The DOE sponsored additional RELAP5 development in the early 1980s to meet its own reactor safety assessment needs. Following the accident at Chernobyl, DOE undertook reassessment of the safety of all of its test and production reactors throughout the United States and the RELAP5 code was chosen as the thermal-hydraulic analysis support tool of choice because of its widespread acceptance [32].

Specific code applications include simulations of transients in light water reactor (LWR) systems such as loss of coolant accidents (LOCAs), anticipated transients without scram, and operational transients such as loss of feedwater, loss of off-site power, station blackout, and turbine trip. RELAP5-3D, the latest in the RELAP5 code series, is highly generic. In addition to calculating the behavior of a reactor coolant system during a transient, the code can be used for simulation of a wide variety of hydraulic and thermal transients in both nuclear and nonnuclear systems involving mixtures of vapor, liquid, noncondensable gases, and nonvolatile solute.

3.3.2 Licensing Use of RELAP5-3D

RELAP5 software was developed as a confirmatory tool rather than a principle enabler for conducting a nuclear plant safety analysis. Consequently, the code is neither enhanced nor controlled to explicitly support the conduct of an independent safety review done in a regulated environment. Licensed code users are required to sign a disclaimer releasing code developers from all legal responsibility for its use in a critical application like an NRC licensing decision. The responsibility to perform the V&Vs and/or specific applications testing necessary to satisfy a regulatory and/or key project objective resides exclusively with the licensed user; the primarily code developers play no formal role in these supplemental qualification efforts undertaken by authorized code users.

Recognizing that RELAP5 codes are not qualified for regulatory use, licensed users have successfully adapted, validated, and used earlier versions of the code to meet NRC licensing requirements related to their specific needs. For example, Siemens Power Corporation (SPC) undertook proprietary V&V development of S-RELAP5 (derived from earlier RELAP5/MOD2 and /MOD3 codes) to satisfy NRC analysis requirements established under 10 CFR 50, Appendix K concerning small-break LOCAs. SPC performed challenging comparison tests and implemented adaptations that increased confidence in code suitability. Meetings were subsequently held between SPC and NRC staff to confirm the adequacy and quality of the SPC code enhancements and application approach. In February 2001, S-RELAP was confirmed as conditionally acceptable for use by the NRC’s Advisory Committee on Reactor Safeguards (ACRS) [33]. Similar proprietary RELAP5 adaptations and V&V efforts were undertaken by other licensed RELAP users (e.g., RELAP5/MOD2-B&W, Framatome Technologies [see NRC ADAMS Accession No. ML030220134]; M-RELAP5, Mitsubishi Heavy Industries [ML093650010]; and N-RELAP5, NuScale Power [ML15299A251]).

To restate this important point, RELAP5 codes were never developed nor are they maintained for use in NRC licensing applications. However, authorized code users have successfully implemented code adaptations on a case-by-case basis and completed the necessary QA activities that allow its use to address regulatory requirements. This represents an alternative approach which may be considered by safety analysis code developers regarding code qualification and use.

3.3.3 Summary of the RELAP5-3D QA Process

RELAP5-3D is maintained under a configuration system that maintains a historical record of all changes. Modification and improvements to coding are reviewed and checked as part of a formal software quality assurance program (SQAP) [34]. The RELAP-3D SQAP summarizes QA activities employed during development and enhancement of the basic RELAP5-3D program. This guidance also establishes the code as meeting "Quality Level 3" software requirements. Quality levels are broadly applied under existing INL procedures to identify unmitigated or potential consequence levels associated with the failure of an item. These levels are assigned to facilitate a common understanding of the rigor to be applied to an item through implementation of appropriate procedures. An INL "Quality Level 3" equates to a "low" level of unmitigated risk associated with the RELAP5-3D Code.

NRC-endorsed standards like NUREG/BR-0167 [5] were not utilized as underlying requirements resource documents for RELAP5-3D. However, all RELAP5-3D software maintenance and modification activities must comply with the scope and requirements set for RELAP5-3D by the SQAP and other applicable INL procedures governing software development.

Assessments, documentation reviews, peer reviews, requirement analysis, and other techniques and methodologies are used for SQA. Most of those activities are based on product reviews, previous experimental results, hand calculations, and associated records. Available automated tools and methods for SQA activities include:

- automated system engineering tools (e.g., requirement management)
- automated test tools (e.g., FORTRAN debugging software
- configuration management tools (e.g., version control, task tracking)
- development tool sets
- build tools (e.g., make utilities, compilers)
- controlled document and records management tools

It is noteworthy that an International RELAP5 Users Group (IRUG) was established and meets once a year to share experiences in RELAP5 development and use. Meeting participants rely on this forum to see and hear about new and proposed features and applications concerning the code and are provided opportunity for input and feedback to developers. Relatedly, a public web page was established to convey IRUG information to other interested stakeholders (http://www4vip.inl.gov/relap5). The web page also provides a convenient venue for public discussions of code licensing, distribution of manuals and newsletters, and offers a standing form by which user problems can be reported directly to developers and subsequently documented.

3.3.4 Components of the RELAP5-3D QA Process

The RELAP5-3D QA process is derived exclusively from existing INL procedures as they relate to the INL "Quality Level 3" standard. The attributes of this standard are identified in the SQAP and encompass multiple activities focused on ensuring a quality level throughout the development cycle appropriate to "low risk" software. These activities are summarized in Table 6.

Table 6. RELAP5-3D software QA tasks

Task	Schedule	Entry criteria	Exit criteria
Risk analysis	As needed	Business requirements.	Approved safety software determination and quality level determination
Management Plan review and approval	As needed	Draft management plan.	Approved by RELAP5-3D asset owner.
Requirements review and approval	Per external release	Draft requirements specification.	Approved by RELAP5-3D asset owner.
Design review and approval	Per external release	Draft design description.	Approved by RELAP5-3D asset owner.
Implementation review	Per release	Baseline software prior to system test.	Documented code walk-through to ensure consistency of software and supporting documentation including traceability of requirements through the life cycle.
System Test	As required by CMP	Completed implementation review.	Approved system test by test case personnel.
Acceptance Test	Per internal release	Completed system test.	Asset approved by RELAP5-3D asset owner.
Problem resolution	As needed	Problem report submitted.	Closed problem report.
In-process QA inspection	Biennially	Initiated by Assurance Portfolio and Integrated Assessment System scheduled start date.	Approved assessment report.

RELAP5-3D was developed to calculate fluid behavioral characteristics during operational and LOCA transients. Even though the code was developed to be confirmatory in nature, some licensed users have successfully “upgraded” the code for (proprietary) regulatory use. Recognizing this, a seven-step process was created to help ensure the code is being appropriately applied and used in the performance of assessment calculations. Although not part of the code qualification process, these seven steps are important to conducting a functional need assessment for RELAP. A summary of this functional use model construction process consists of:

1. Transient scenarios should first be evaluated from a perspective of whether the code has the capability to analyze the expected phenomena.
2. The information required to build the model must be collected. This information consists of system geometry specifications and system initial and boundary conditions.
3. Information that describes the hardware as well as the hardware initial and boundary conditions must be “translated” to the form required by RELAP5-3D.
4. Nodalization resulting from the above process should be reviewed by a model review committee before an analysis is performed. The committee reviews the important phenomena that will occur during the transient and determines whether the model and planned analysis approach will be adequate to evaluate transient behavior and meet analysis objectives.
5. The steady-state calculation is performed and analyzed. The analyst must ensure that the model’s initial condition is representative of the real system’s condition.
6. The transient calculation is performed and analyzed. During this phase of the analysis process, the analyst must ensure that code results are representative of the subject transient. Unphysical results

caused by improper nodalization, code deficiencies, or user errors must be identified and eliminated. Thereafter, the analyst can use results to meet desired analysis objectives.

7. Throughout the process, the analysis must be rigorously documented. The model should be documented in a workbook and independently checked, when feasible, by another analyst. The calculation should be outlined, the steps taken to ensure that the calculation is representative of the subject transient should be listed, and analysis results should be recorded.

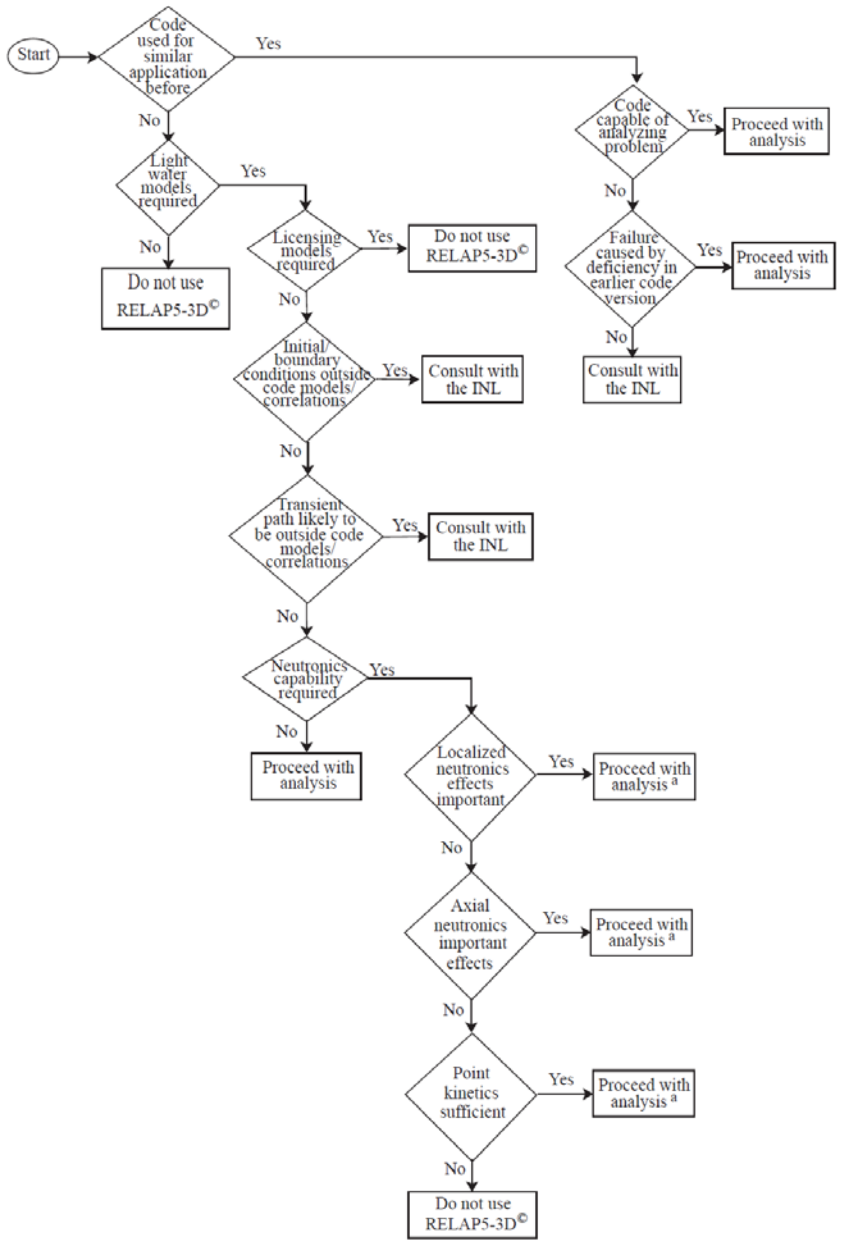
The results of a RELAP5-3D analysis can be used to provide a basis for calculating structural loads and conducting water or steam hammer analysis. However, subsequent analyses of such systems were not included in RELAP5 development plans and the numerical techniques have not been optimized for such applications. A rigorous assessment of code results has not been conducted on these specific topics. In response to this situation, users are further alerted to this constraint through another functional needs analysis aid that is provided in user documentation [35]. This functional aid is illustrated in Fig. 6.

3.3.5 RELAP-3D Testing, V&V

Theory and implementation of code improvements are validated through assessment calculations that compare code-predicted results to idealized test cases or experimental results. In 2012, it was identified that recent code development tasks expanded the requirements for testing proposed code modifications [36]. The testing needed to be more extensive, systematic, and rigorous than what was done in the past. As noted during the 2014 IRUG meeting, the new RELAP5-3D verification testing that was developed evaluated 194 code features. This testing included 43 test problems with 125 input cases [37]. Comparing verification files for the same input reveals changes between code versions or application of code capability. The new verification capability was used to locate code problems with:

- unexpected calculation changes going from version to version,
- restart issues,
- backup issues,
- multi-case issues, or
- parallel virtual machine (PVM) coupling issues

Backup issues" are revealed when a base run is compared to a run that repeats every time-step. "Multi-case issues" arise when the code fails when fed a multi-case input but runs properly when each input case is run separately. "PVM issues" refers to verification testing failures that were caused by code timing errors that caused PVM synchronously-coupled installation problems to fail. All issues uncovered with the original verification test suite were solved. Additional verification capabilities are also being developed.



Start
Code used for similar application before
Yes
No
Code capable of analyzing problem
Yes
Proceed with analysis
No
Failure caused by deficiency in earlier code version
Yes
Proceed with analysis
No
Consult with the INL
Light water models required
Yes
No
Do not use RELAP5-3D©
Licensing models required
Yes
Do not use RELAP5-3D©
No
Initial/ boundary conditions outside code models/ correlations
Yes
Consult with the INL
No
Transient path likely to be outside code models/ correlations
Yes
Consult with the INL
No
Neutronics capability required
Yes
No
Proceed with analysis
Localized neutronics effects important
Yes
Proceed with analysis a
No
Axial neutronics important effects
Yes
Proceed with analysis a
No
Point kinetics sufficient
Yes
Proceed with analysis a
No
Do not use RELAP5-3D©

a. If a user intends to conduct a structural, water or steam hammer analysis using RELAP5-3D, consult section 2.1.2.

Fig. 6. RELAP5-3D functional use analysis.

3.3.6 Lessons Learned

It is essential to note that RELAP codes never explicitly incorporated the software requirements established and/or formally endorsed by the NRC. However, RELAP codes are used to address NRC regulatory requirements through the quality assurance efforts done by individual code users. As such, this process illustrates an alternative pathway for potentially qualifying a code for regulatory use, (i.e., the basic code is developed, maintained, and shared with licensed code users who then bear full and complete responsibility for performing appropriate V&V of the code for their specific regulatory purposes). However, given the V&V actions performed by an individual licensed code user typically remains proprietary for that individual user, a paradigm is established that may not be as efficient nor cost effective as it could be with respect to code use by the entire regulated community.

Despite this approach, however, user-modified RELAP5 codes have been used extensively in NRC safety decisions. It is up to the individual code developer to decide if their end use objectives can and should be functionalized in a way similar to what is used in RELAP5 codes. In any event, NRC reviews of user-adapted RELAP codes have yielded important insights worth noting by all code developers. Some of these lessons were communicated by the ACRS with respect to RELAP5 codes [38]:

- "Good documentation is sound quality control practice. It provides insurance against costly delays, uncertainties, confusion, and mistakes. It simplifies and enhances staff and ACRS reviews and aids users. It also builds confidence in the soundness of regulatory judgments in the broader technical community."
- "In the future, the staff should insist on complete documentation before issuing a final SER."
- "The staff needs to consider how broad-based the assessment of realistic codes should be, not only to ensure adequacy but also to measure uncertainty. Clear criteria are also needed on what constitutes an adequate database for assessing this uncertainty and on how this should be done quantitatively."
- "SPC provided the staff with a working version of their code and input decks to enable test conditions to be simulated. However, the staff informed us that it had not run the code as an independent check, nor used this capability to investigate some key features. We understand that the staff's rationale in this particular case is that it is familiar with previous relevant applications of RELAP5. The use of the codes by the staff should be an important part of its review process. We look forward to staff reports on its independent evaluation of code runs when S-RELAP5 is submitted as a realistic code."
- "Because we cannot check many features of a complex code, some of our assessment must be based on establishing confidence in the applicant's technical judgment. In the present case, we have been helped in our evaluation by the cooperation of SPC in responding to our technical questions and supplying additional information. Another important factor in establishing confidence is the provision of accurate, complete, and unequivocal documentation."

3.4 TRACE

This subsection is focuses on conveying the highlights of the TRACE best-estimate reactor systems code QA program. The TRACE software is a collection of legacy codes that were developed as confirmatory analysis tools. Therefore, the discussion of RELAP5-3D in Section 3.2.2 is also applicable partially applicable to the TRACE code.

3.4.1 Introduction to TRACE

TRACE is a best-estimate system code designed to predict steady-state and transient thermal hydraulic behavior in LWRs. TRACE is particularly relevant to SAS4A/SASSYS because it is a safety-related systems code. Additionally, TRACE is relevant because it is the integration of the capabilities of four legacy NRC codes: TRAC-P, TRAC-B, RELAP5, and RAMONA). The integration of these four legacy codes into one modern product was a significant challenge in the development, validation, and assessment of the TRACE code [38].

TRACE is composed of modernized versions of legacy tools with similar capabilities. The development of TRACE required integration of the capabilities unique to specific legacy tools, for example the CHAN (channel) component in TRAC-B and side junction components in RELAP5 [38].

3.4.2 Summary of the TRACE QA Process

The integration of legacy tools into the TRACE platform also included significant enhancements in code QA. The QA process is complex given that TRAC-P, TRAC-B, RAMONA, and RELAP-5 were application-specific tools, and that TRACE is a platform integrating the different application space and functional needs of all of these tools. Additionally, the multidimensional nodal diffusion, transport, and reactor kinetics tool PARCS is coupled to TRACE. TRACE also contains enhanced models for gap conductance from the NRC fuel performance tool, FRAPCON. In this sense, TRACE is similar to SAS4A/SASSYS, which combines many modules with completely different functional needs and associated physics.

The development of TRACE also introduced new physics to several of the tools necessary to analyze advanced LWR concepts and passive safety features. These capabilities were initiated to support design certification reviews for new reactors.

The NRC approach to code QA is a multiple-step process [38]:

1. Verification is an important step, and includes unit and regression testing to ensure that the functionalities of TRACE perform as expected. This includes a rigorous configuration management process. Changes to the code are not permitted until they pass the regression suite of test cases and are documented and approved by the custodian of the code. More than 2500 regression tests are used for TRACE.
2. Validation is another major step in the TRACE QA process. Validation in TRACE focuses on a three-level hierarchy: fundamental problems, separate effects tests, and integral effects tests. Fundamental problems are very basic phenomena and are closely related to tests in the regression test suite. The CSAU methodology is applied in the more detailed assessment cases that make up the separate effects tests and integral effects tests.

3.4.3 Components of the TRACE QA Process

Adherence to the TRACE QA guidelines, which are congruent with NUREG/BR-0167 requirements for configuration management, is the first step in the TRACE verification process. The second step, the TRACE regression test suite, includes tests that range from valve-closing logic to verification of individual models or correlations [38]. Example fundamental assessment cases are shown Table 7.

The TRACE Code Assessment and Maintenance Program (CAMP) draws from a large set of NUREG reports [38]. The CAMP approach spans the entire parameter space of accident and transient analyses for the TRACE code [38]. The TRACE Developmental Assessment Manual identifies a number of separate and integral effects tests. In addition, the TRACE CAMP uses separate effects tests that assess [38]: reflood heat transfer, break flow, steam generator hydraulics, loop seal clearance, and flooding at the upper core plate. Integral behavior tests are also a vital component of the TRACE QA process.

Table 7. Fundamental validation assessment cases for TRACE, from Reference 38

Fundamental assessment case	Cases	Purpose
Radial and Axial Heat Conduction	2	Compare heat conduction calculation to exact solutions.
Drain - Fill	4	Examine ability to track water level across node boundaries.
Oscillating Manometer	1	Compare calculation of two-phase interface to exact solution.
ANL Vertical Two-Phase Flow	71	Prediction of void fraction in vertical two-phase upflow.
Two-phase flow test/horizontal flow tests	110	Prediction of two-phase flow in a large diameter horizontal pipe.
Single and Two-Phase Wall Friction	27	Prediction of wall friction component of pressure drop.
Single Tube Flooding	3	Examine calculation of flooding and counter current flow limitation correlations.
Adiabatic Tube	1	Assess interfacial shear under adiabatic conditions.

3.4.4 TRACE Testing, Validation, and Verification

Central to the NRC validation approach is the CSAU methodology. According to Reference 38:

> CSAU provides a systematic approach to the application of a systems code to a large scale facility subjected to a complex transient scenario. This is where separate effects tests and integral tests are vital. The methodology is structured and is sufficiently general so that it can be applied to a wide variety of plant designs. The overall framework of CSAU is designed to address three important questions regarding a code to be used for nuclear power plant safety calculations:
>
> 1. Has the code the capability to scale up phenomena observed in small-scale test facilities to full- size nuclear power plants (NPPs)?
> 2. Can the code be applied to safety studies of a particular scenario or a set of scenarios for a given plant design?
> 3. What is the uncertainty with which the code calculates important parameters, say the peak cladding temperature, in a full scale NPP?
>
> TRACE V&V makes particular use of the first two Elements of CSAU; Requirements and Capabilities, and Assessment and Ranging of Parameters. Code requirements depend on the scenario, and the models necessary to simulate the scenario. Assessment requires the comparison of code performance against experimental data to determine potential code limitations. Central to these two steps is development of a Phenomena Identification and Ranking Table (PIRT) to identify those physical processes most important to successful simulation of the scenario. For TRACE assessment, three PIRTs were used to identify processes of "generic" interest to large and small break LOCAs.

An example of the NRC code development and assessment process flow chart is shown in Fig. 7.

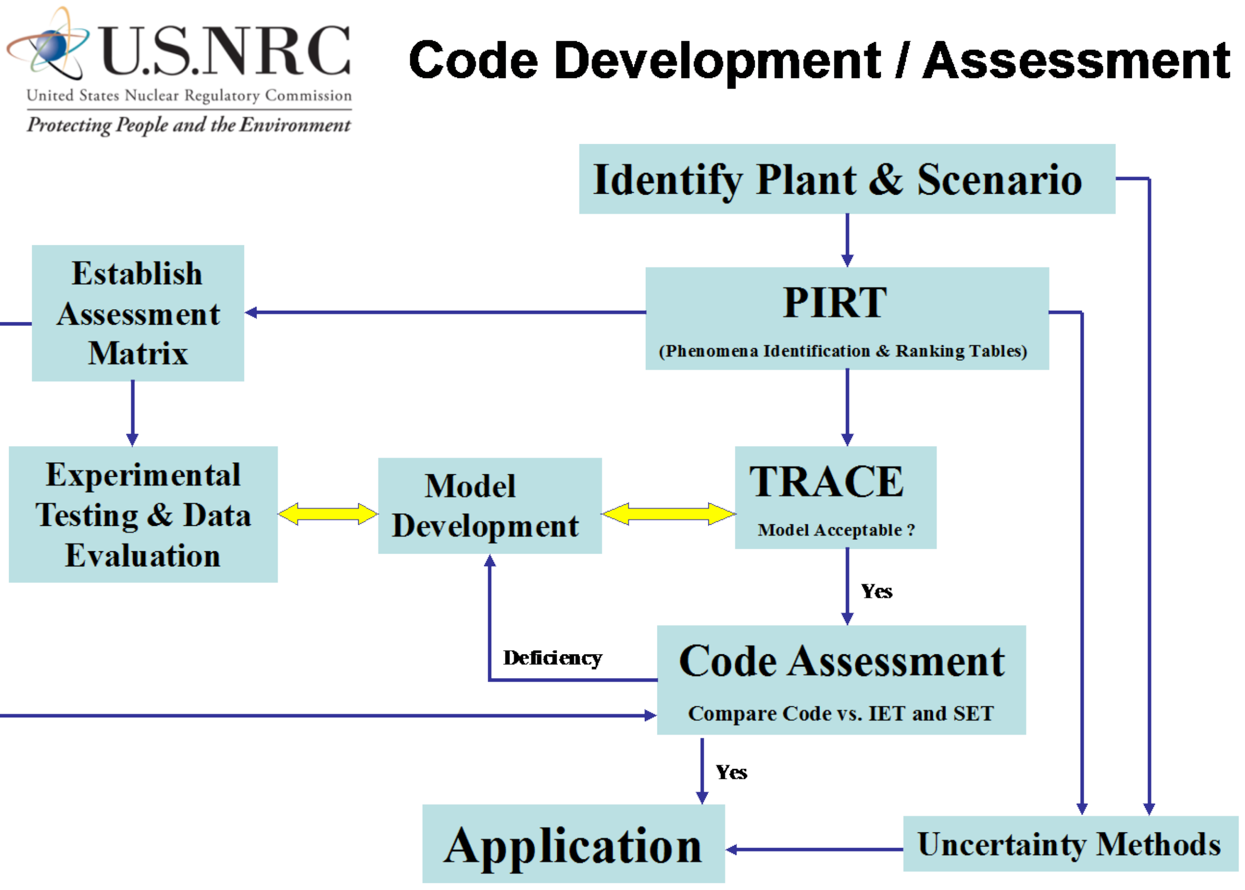

U.S.NRC
United States Nuclear Regulatory Commission
Protecting People and the Environment
Code Development / Assessment
Identify Plant & Scenario
Establish Assessment Matrix
PIRT
(Phenomena Identification & Ranking Tables)
Experimental Testing & Data Evaluation
Model Development
TRACE
Model Acceptable ?
Yes
Deficiency
Code Assessment
Compare Code vs. IET and SET
Yes
Application
Uncertainty Methods

Fig. 7. NRC code development and assessment framework from Reference 39.

An overview of the NRC regression suite for TRACE verification is shown in Table 8.

Table 8. NRC regression suite for TRACE verification from Reference 38.

Regression test category	Number of cases
Numeric and solution procedures	1296
Input and output	301
Control systems	144
Power and kinetics	225
Flow process models	232
Closure models	301
Heat structures	982
Integral behavior	318
Component models	1336

3.4.5 Lessons Learned and Path Forward

The TRACE QA process relies heavily on a rigorous configuration management process, a library of over 2500 regression cases, and the underlying framework of the CSAU methodology. TRACE is highly applicable to SAS4A/SASSYS, primarily because it is an integration of a large number of reactor-safety-related modules with different functionalities and representing different physics.

The TRACE QA process is a highly relevant example of an application of the CSAU methodology. The emphasis on a number of PIRTs (analogous to functional needs assessments) also serves as an ideal example of how to identify the physical processes and associated code functionalities relevant to test in important scenarios, such as small- and large-break LOCA.

Another specific example is the recent extension of TRACE to support design certification of advanced light water reactor designs. The systematic approach serves as a model for the application of the CSAU methodology to SAS4A/SASSYS QA [38]:

> An independent PIRT is first developed, and then the TRACE models and correlations are reviewed for applicability. The PIRT is developed as early as possible in the licensing review, in order to allow as much time as possible in the schedule for model development and independent testing (if necessary). Specific assessment is performed and included in an applicability report, which serves to document the PIRT and evaluation of TRACE models for the particular design.

3.5 MPACT (CASL)

This subsection is focused on conveying the highlights of the Michigan Parallel Characteristics Transport (MPACT) Code reactor neutronics code QA plan.

3.5.1 Introduction to MPACT

The MPACT code provides the capability to perform high fidelity deterministic simulations of the neutron distribution for pressurized water reactors including coupling to depletion capability and analysis of multiple fuel cycles [40]. This subsection is intended to provide an explicit example of an implementation of the CASL SQAP [41]. MPACT has a rigorous library of unit testing and regression tests and is an example of a modern tool that has been developed in an era of enhanced software QA, as opposed to a legacy tool where elements of the QA process were retroactively developed after the tool.

3.5.2 Summary of the MPACT QA Process

The MPACT QA process is built on the foundation of a rigorous library of unit tests and regression tests. Approximately 80% of MPACT code is covered by unit and regression tests [40]. Because the MPACT V&V process was developed concurrently with the development of the tool, the unit testing process was instrumental in finding and resolving problems early [40]. Additionally, the complexity of the physics problems solved in MPACT yields challenges [40]: “One of the challenges in writing the unit tests within MPACT has been the difficulty of setting up realistic tests with relevant initial conditions so the part of the application being tested behaves like part of the complete system. If these initial conditions are not set correctly, the test will not be exercising the code in a realistic context, which diminishes the value and accuracy of unit test results.” Regression testing also plays a key role in the MPACT QA process.

The role of testing in the MPACT QA process is fully integrated into the development of the code. For example, the key components of the MPACT automated testing process are as follows [40]:

1. Test Server checks for changes every 10 minutes and tests two configurations;
2. Tests many more regression tests, performed by CASL and UM test machines;
3. Test GNU C Compiler 4.6.1, 4.7.2, 4.8.1, Intel 12.1.5 with and without message passing interface and other task parallel libraries;
4. Unit tests for solver kernels test against analytic solutions. Some regression tests compare against analytic solutions;
5. Depletion solver is compared to experimental results;

6. This means analyzing program with Valgrind; and
7. This means running ‘gcov’ on all tests.

3.5.3 MPACT Testing, Validation and Verification

Because MPACT is both a standalone code and also a module in the larger CASL Virtual Environment for Reactor Applications Core Simulator (VERA-CS) suite, the testing suite is very focused on single physics (reactor physics) testing. The MPACT code emphasizes unit tests strongly during the development process. For example [40],

> The overall goal of unit testing is to isolate each part of the program and show that the individual parts are correct. The testing in MPACT was designed to verify the smallest testable part of an application and each test case was designed to be independent from the others. The practice in MPACT has been for developers to create unit tests for all functions and methods while the code itself is being written. When the tests pass, that phase of the code development is considered complete. However, if a unit test fails, there is considered to be a bug either in the changed code or the tests themselves, and that phase of the code development process is continued.”

A summary of the unit and regression test coverage in the MPACT code is shown in Table 9. The emphasis and tight coverage of the MPACT unit testing in particular has led to identification of many bugs very early in the development cycle [40]. In Table 9 the coverage of unit tests refers to the percentage of the lines of code that are subjected to unit testing.

Table 9. Unit and regression test coverage in MPACT from Reference 40

Metric	Libraries/subroutines	Code drivers	Executable	Total
Unit tests	126	4	0	130
Regression tests	0	0	66	66
Coverage (%)	79.60	60.15	62.90	78.90
Lines of code	108,101	3,784	385	112,270

Regression or integral testing also plays a key role in the MPACT V&V process. Regression testing is primarily used to verify key features in integrated simulations, such as geometry, transport solvers in 2D, transport solvers in 3D, other solvers, and parallel processing. The building blocks of the MPACT features that are tested in each of these areas are shown in Table 10.

Table 10. Building blocks of core MPACT features covered by regression testing, adapted from Reference 40

Geometry	Transport solvers	Other coupled solvers	Parallel processing
Cylindrical, quarter, rectangular and generalize cylinder pin geometries	2D method of characteristics	Depletion (native and Origen)	Message passing interface (space, angle, space+angle), explicit file
Inserts	2D-1D with simplified transport	Search (boron, rod)	OpenMP (threading)
Control rod (+ rod movement)	P0 and Pn 2D-1D with nodal expansion method	Multistate	

Table 10. Building blocks of core MPACT features covered by regression testing, adapted from Reference 36 (continued)

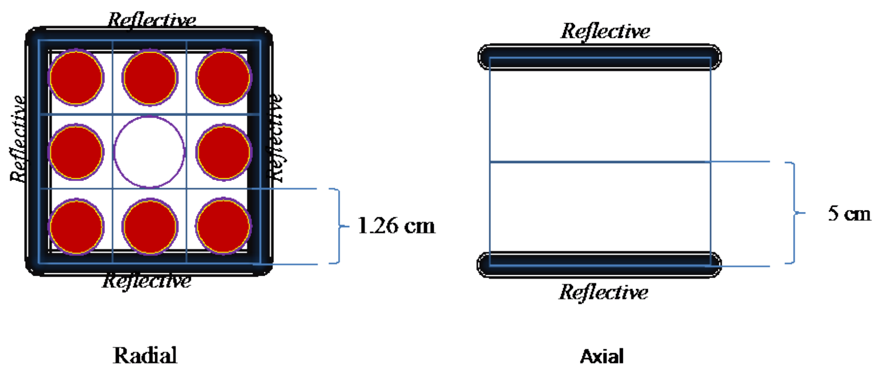
Geometry	Transport solvers	Other coupled solvers	Parallel processing
Baffle/reflector		Coarse mesh finite difference	
Upper/lower nozzle, core plate, reflector		Thermal hydraulic Feedback (internal and external)	
Multiple assemblies/modules		Equilibrium fission product poisons	
Symmetry		Cross section shielding (subgroup vs embedded self-shielding method)	
Grids		Control rod cusping	
Detectors			

An example MPACT unit test is focused on a mono-energetic problem with an angular surface flux boundary condition fixed in a purely absorbing media. Because the problem has an analytical solution ($\Psi^{out} = \Psi^{in} e^{-\Sigma_t s}$), it represents an ideal unit test. Reference 40 describes the test in further detail:

> The focus of this unit test is the Product Quadrature sweeper module of the MOC [method of characteristics] solver in the code which loops through angles in the azimuthal quadrature set which is the ‘Chebyshev’ quadrature for this problem. For each angle in the azimuthal quadrature set the long rays are swept and modular rays are looped through for each long ray. For each modular ray the angles are swept in the polar quadrature set which is the Gauss quadrature for this problem..

The code snippet shown in Appendix B, which represents important aspects of this test, includes the specification of boundary conditions and the fixed value of the angular surface flux (2.0_SRK).

An example MPACT regression test yields insight into integral testing of the code features (see Fig. 8). This example focuses on solving a 3-D problem with reflective boundary conditions on each surface.



0.99180	1.00820	0.99180
1.00820	0.00000	1.00820
0.99180	1.00820	0.99180

Fig. 8. Example regression test and solution from Reference 40.

3.5.4 Lessons Learned and Path Forward

MPACT is an idealized example of a modern tool developed synergistically with a sophisticated V&V approach. MPACT is a single-purpose, single physics tool (although it uses multiple methods and solves different problems). In many ways, MPACT is representative of a single physics “module” within SAS4A/SASSYS.

The key lesson learned is that integrating a rigorous regression suite into the development process significantly enhances the ability to identify and eliminate code bugs or other problems. Another key lesson is that the unit test suite, and the expert elicitation that is used to develop the test suite, must be focused on developing realistic tests that are representative of the way the code is actually used. Additionally, the tests must replicate how the code behaves as part of the complete system (both MPACT and within the larger CASL VERA-CS suite).

4. RECOMMENDATIONS FOR SFR SAFETY ANALYSIS CODE QUALIFICATION

The objective of this section is to provide recommendations for implementing a SAS4A/SASSYS QA plan and building a documented pedigree to support its use in NRC licensed applications.

The NQA-1-2008/2009 standard requires that software not produced under a QA program compliant with the standard be dedicated in accordance with the requirements of Part II, Subpart 2.14, *Quality Assurance Requirements for Commercial Grade Items and Services*, and licensees are likely to attempt to use the commercial-grade dedication process to qualify SAS4A/SASSYS for safety-related analyses; a major goal should be to develop the documentation needed to support this process.

The recommendations can be summarized in three objectives:

1) Evaluate the QA applied to the code during its development, the suitability of existing documentation, and identify limitations, vulnerabilities, and areas for improvement and the resources needed to address them.
2) Establish a NQA-1-2008/2009 compliant SQA program for future maintenance and development.
3) Develop the documentation base needed to support commercial-grade dedication.

These objectives are further elaborated upon in the following sections.

4.1 EVALUATION

This process is a prerequisite step for software improvement. It will provide documented evidence to potential licensees of the current limitations and vulnerabilities of the code as well as help define and prioritize the steps required for improvement. The software should be placed under configuration control prior to the evaluation.

In brief, the evaluation should [9]:

a) determine the adequacy of the software documentation to support testing, operation, and maintenance.
b) identify activities to be performed throughout the applicable lifecycle of the software including preparation of required documentation and performance of required reviews and/or tests,
c) determine the software's capabilities and limitations for intended use,
d) specify test plans and test cases required to validate the capabilities within the stated limitations,
e) identify instructions for software use within the limits of its capabilities,
f) identify any exceptions to the lifecycle documentation and its justification.

We recommend adapting the plan and methods discussed in DOE-EH-4.2.1.2-Criteria [9], Section 2, to perform the evaluation. Table 11 below reproduces the evaluation plan from this reference.

Table 11. Plan for SQA Evaluation of existing safety analysis software (Reproduced from DOE-EH-4.2.1.2-Criteria Table 2-2)

Prerequisites	a. Determine that sufficient information is provided by the software developer to allow it be properly classified for its intended end-use. b. Review SQAP per applicable requirements in Table 3-3.
Software Engineering Process Requirements	a. Review SQAP for: a. Required activities, documents, and deliverables b. Level and extent of reviews and approvals, including internal and independent review. Confirm that actions and deliverables (as specified in the SQAP) have been completed and are adequate. b. Review engineering documentation identified in the SQAP, e.g., a. Software Requirements Document b. Software Design Document c. Test Case Description and Report d. Software Configuration and Control Document e. Error Notification and Corrective Action Report, and f. User's Instructions (alternatively, a User's Manual), Model Description (if this information has not already been covered). c. Identify documents that are acceptable from SQA perspective. Note inadequate documents as appropriate.

Table 11. Plan for SQA Evaluation of existing safety analysis software (Reproduced from DOE-EH-4.2.1.2-Criteria Table 2-2) (continued)

Software Product Technical/Functional Requirements	a. Review requirements documentation to determine if requirements support intended use in Safety Analysis. Document this determination in gap analysis document. b. Review previously conducted software testing to verify that it sufficiently demonstrated software performance required by Software Requirements Document. Document this determination in the gap analysis document.
Testing	a. Determine whether past software testing for the software being evaluated provides adequate assurance that software product/technical requirements have been met. Obtain documentation of this determination. Document this determination in the gap analysis report. b. (Optional) Recommend test plans/cases/acceptance criteria as needed per the SQAP if testing not performed or incomplete.
New Software Baseline	a. Recommend remedial actions for upgrading software documents that constitute baseline for software. Recommendations can include complete revision or providing new documentation. A complete list of baseline documents includes: a. Software Quality Assurance Plan b. Software Requirements Document c. Software Design Document d. Test Case Description and Report e. Software Configuration and Control f. Error Notification and Corrective Action Report, and g. User's Instructions (alternatively, a User's Manual) b. Provide recommendation for central registry as to minimum set of SQA documents to constitute new baseline per the SQAP.
Training	a. Identify current training programs provided by developer. b. Determine applicability of training for DOE facility safety analysis.
Software Engineering Planning	a. Identify planned improvements of software to comply with SQA requirements. b. Determine software modifications planned by developer. c. Provide recommendations from user community. d. Estimate resources required to upgrade software.

4.2 NQA-1-2008/2009 COMPLIANT SQA PROGRAM

One of the key steps necessary to implement NRC-acceptable QA requirements is to develop a comprehensive QA plan including a configuration management plan and an error reporting process. The SAS4A/SASSYS QA plan should be mapped to NQA-1-2008/2008 Part II Subpart 2.7 QA requirements and other relevant standards, if deemed applicable. A standardized configuration management plan is also a necessary step to enhance the overall QA approach of the SAS4A/SASSYS code system and to fully implement the QA plan. A modern configuration management and change tracking system is a key enhancement. A standardized system for tracking and managing bug reports from users and associated documentation of discrepancies is recommended to meet NRC nonconformance and corrective action requirements. Although not explicitly NQA-1-2008/2009 compliant, the SCALE QA [21] and configuration management [20] plans are excellent examples and could be refined for use with SAS4A/SASSYS. The guidance provided by NUREG/BR-0167 [5] is also relevant and provides a good roadmap to practical implementation of SQA requirements.

The following IEEE software development standards may also be used to help meet the requirements of NQA-1-2008/2009 Part II Subpart 2.7:

- IEEE 730 Software Quality Assurance Plans
- IEEE 828 Software Configuration Management Plans

- IEEE 829 Software and System Test Plans
- IEEE 830 Software Requirements Specification
- IEEE 1012 Software V&V Plans
- IEEE 1058 Software Project Management Plans
- IEEE 1228 Software Safety Plans

A very useful document is NUREG-1737 [6], which outlines the NRC QA procedures for thermal hydraulic codes. NUREG-1737 includes a set of checklists that are very useful as a procedural map to establishing software QA that is complaint with NUREG/BR-0167 [5]. These example checklists are reproduced in Appendix C. It is important to note that these checklists are helpful as general guidance but are not sufficient for commercial-grade dedication of a particular tool.

The plans and procedures defined in DOE-EH-4.2.1.2-Criteria Section 3 and Table 3-3 for demonstrating NQA-1 compliance are very useful, and can be adapted for use by SAS4A/SASSYS.

A particular challenge will be the development of a software requirements specification, if one does not already exist. The requirements specification consists of functional needs, performance, design constraints, attributes, and external interfaces. To meet the requirements of NQA-1, a software requirements specification document must be generated.

The first emphasis of the requirements definition is a functional needs assessment. A functional needs assessment determines what the code should do and the range of its applicability. Functional needs encompass range of applicability, scalability of the models to reactor plant applications, assessment base, and accuracy. The functional needs assessment will identify the specific features of each module that will be subject to testing. This will inform a further investigation into the specific subroutines within the modules that directly feed features.

The performance requirements definition includes time-related issues, numerical accuracy issues and acceptance criteria, and scalability [6]. Performance requirements include definition of a qualification test plan encompassing [6]:

a. The number and types of qualification problems to be completed,
b. The rationale for their choice, why was this problem chosen, which functional requirement does it test?
c. The specific range of parameters and boundary conditions for which successful execution of the problem set will qualify the code to meet specific functional requirements,
d. Significant features not to be tested and the reasons (for example, for complex codes, absolute qualification of every combination of options over every usable range of parameters is not practical)
e. Acceptance criteria for each item to be tested. Number and types of sensitivity calculations to be performed in order to develop user guidelines.
f. Discussion of scalableness [sic], if applicable.

Development of a rigorous V&V test suite is also a part of meeting NRC QA requirements. This includes a suite of unit tests. One potential approach is to develop comprehensive tests for each class and method in an object-oriented programming paradigm. In a procedural programming paradigm, at least one unit test per subroutine or functionality is appropriate. The experience of MPACT is relevant, where setting up realistic tests with relevant initial conditions was identified as a key challenge. It is vital that the unit test suite provides realistic tests, and that those tests be mapped to documented requirements. If any functionality is discussed in the user manual, it should have at least one unit test associated with it. A Test Report or Verification & Validation manual should present a cross-index of requirements to corresponding tests.

Unit test development could be “bootstrapped” via a similar method as the modern SCALE regression test suite. This would include assembling a group of “expert users” with knowledge in their relevant specializations, and focusing this team of experts on assessing each module within SAS4A/SASSYS. One challenge here could be the possible base of experts, as there are relatively few groups within the United States working on fast reactor safety analysis. Potential opportunities include students from universities that have used or are using the SAS4A/SASSYS codes. Additionally, use of SAS4A/SASSYS internationally could be leveraged to develop input to potential unit testing. A graded approach is recommended, bearing in mind the relative importance of a particular module or result.

Integrating a rigorous regression suite into the development process significantly enhances the ability to identify and eliminate code bugs or other problems. The unit test suite must be focused on developing realistic tests that are representative of the way the code is used. Ultimately, the unit test development will require a rigorous functional needs assessment for each of the modules in SAS4A/SASSYS. The goal of the regression suite should be to have a library of realistic tests that cover the most important functionalities of the tool. Examples of well-covered regression suites include MPACT (approximately 80% coverage of classes and functionalities) and the SCALE module ORIGEN (approximately 70% coverage of classes and functionalities).

Integral testing is another need, including tests of the coupling between the different modules of SAS4A/SASSYS. Tests would include both physics-based integral tests developed from expert elicitation and benchmark tests that are based on actual plant data (for example, from the Experimental Breeder Reactor II).

Another broader issue is the available data for model validation and code verification. Data used to develop the applicable physics models cannot be used to independently validate the models or to verify the tools. The TRACE QA process is a highly relevant example of the application of the CSAU methodology. In general, TRACE is an excellent example of the approach that could be followed for SAS4A/SASSYS, given that it represents an integration and modernization of four legacy tools.

4.3 SUPPORT FOR COMMERCIAL-GRADE DEDICATION

It is vital to begin to build a base of documentation that can be used to support the commercial-grade dedication process. For this activity the guidance provided in EPRI Technical Report 3002002289 [3] should be followed. A good start would be to begin to generate documented evidence appropriate to address the critical characteristics identified in Section 6, Tables 6-2 through 6-6, of this report.

Only those software requirements relevant to critical characteristics are important for the dedication process, and this can greatly reduce the scope of testing, verification, and validation necessary to perform. The critical characteristics must be determined from the detailed needs of the simulation(s) to be performed for licensing purposes. It is important to establish relationships with potential licensees to develop lists of critical characteristics for meaningful use cases, including output figures of merit that are used to make or inform safety decisions. From the figures of merit, the relevant models and associated software requirements can be identified using a PIRT analysis.

Development of a framework and platform for an international SAS4A/SASSYS user community is a key recommended action. The more users are exercising a tool, the better the quality of the tool will become, because the users will report bugs. For example, a key source of bug reports in SCALE is bug reports from the user community. SCALE also has mailing lists and an on-line user community where users can discuss various problems in a collaborative way. Replicating this end-user community and experience with SAS4A/SASSYS would enhance QA. The user base for SAS4A/SASSYS is much smaller than it is for SCALE, but an active user community would also act to grow the user base. Many regression tests in SCALE are based on user feedback from training/sample problems and bug reports. It is vital to build an open community among existing SAS4A/SASSYS users for information exchange.

It is important to note that the commercial-grade dedication approach is most relevant if a license applicant will use SAS4A/SASSYS. If the intended application of SAS4A/SASSYS is as a confirmatory tool utilized by the NRC, the approach outlined in NUREG/BR-0167 [5] is most relevant.

5. REFERENCES

1. U.S. Nuclear Regulatory Commission, *Quality Assurance Program Criteria (Design and Construction)*, Regulatory Guide 1.28, Revision 4, Office of Nuclear Regulatory Research, Washington, DC, June 2010.
2. American Society of Mechanical Engineers (ASME), *Quality Assurance Requirements for Nuclear Facility Applications*, NQA-1-2008, ASME International, New York, NY, March 2008.
3. M. Tannenbaum and M. Tulay, *Plant Engineering: Guideline for the Acceptance of Commercial-Grade Design and Analysis Computer Programs Used in Nuclear Safety-Related Applications*, Electric Power Research Institute, 2013 Technical Report 3002002289 (Rev 1 of 1025243), 2013.
4. S. Birla, R. Sydnor, and N. Carte, *(Availability of) An International Report on Safety Critical Software for Nuclear Reactors by the Regulator Task Force on Safety Critical Software (TF-SCS)*, NUREG/IA-0463, Division of Engineering, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, Washington, DC, December 2015.
5. U.S. Nuclear Regulatory Commission, *Software Quality Assurance Program and Guidelines*, NUREG/BR-0167, Division of Information Support Services, Office of Information Resources Management, Washington, D.C., February 1993.
6. U.S. Nuclear Regulatory Commission, *Software Quality Assurance Procedures for NRC Thermal Hydraulic Codes*, NUREG-1737, Division of Systems Analysis and Regulatory Effectiveness, Office of Nuclear Regulatory Research, Washington, DC, November 2000.
7. U.S. Department of Energy, DOE Order 414.1D, *Quality Assurance*, Washington, DC, April 25, 2011.
8. U.S. Department of Energy, DOE Guide 414.1-4, *Safety Software Guide for Use with 10 CFR 830 Subpart A, Quality Assurance Requirements and DOE O 414.1D, Quality Assurance*, Washington, DC, June 17, 2005.
9. U.S. Department of Energy, *Software Quality Assurance Plan and Criteria for the Safety Analysis Toolbox Codes*, DOE-EH-4.2.1.2-Criteria, Office of Environment, Safety and Health, Washington, DC, November 2003.
10. U.S. Department of Energy, *Fuel Cycle Technologies Quality Assurance Program Document, Revision 2*, Office of Nuclear Energy, December 20, 2012.
11. U.S. Department of Energy, Nuclear Energy Advanced Modeling and Simulation (NEAMS) Program Software Quality Assurance Plan (NSQAP) Revision 1.6, Lawrence Livermore National Laboratory Technical Report, LLNL-SM-455533, April 11, 2013.
12. NUREG/CR-5249, *Quantifying Reactor Safety Margins, Applications of Code Scaling, Applicability, and Uncertainty Evaluation Methodology to a Large-Break, Loss-of-Coolant Accident*, EGG-2552 R4, Idaho National Laboratory, U.S. Department of Energy, October 1989.
13. Consortium for Advanced Simulation of LWRs, *Quality Manual*, CASL-U-2012-0047-000, U. S. Department of Energy, Office of Nuclear Energy, Washington, DC, March 30, 2012.
14. American Society of Mechanical Engineers (ASME), *Addenda to ASME NQA-1-2008: Quality Assurance Requirements for Nuclear Facility Applications*, NQA-1a-2009, ASME International, New York, NY, August 2009.

15. Institute of Electrical and Electronics Engineers (IEEE), *IEEE Standard for Software Safety Plans*, IEEE Std. 1228-1994, Software Engineering Standards Committee of the IEEE Computer Society, New York, NY, March 17, 1994.
16. Institute of Electrical and Electronics Engineers (IEEE), *IEEE Standard for Software Project Management Plans*, IEEE Std. 1058-1998, Software Engineering Standards Committee of the IEEE Computer Society, New York, NY, December 1998.
17. Institute of Electrical and Electronics Engineers (IEEE), *IEEE Standard for Configuration Management is Systems and Software Engineering*, IEEE Std. 828-2005, Software Engineering Standards Committee of the IEEE Computer Society, New York, NY, 2005.
18. Institute of Electrical and Electronics Engineers (IEEE), *IEEE Standard for Software QA Processes*, IEEE Std. 730-2002, Software Engineering Standards Committee of the IEEE Computer Society, New York, NY, 2002.
19. Institute of Electrical and Electronics Engineers (IEEE), *IEEE Standard for Software and System Test Documentation*, IEEE Std. 829- 1998, Software Engineering Standards Committee of the IEEE Computer Society, New York, NY, 1998.
20. Institute of Electrical and Electronics Engineers (IEEE), *IEEE Standard for System and Software V&V*, IEEE Std. 1012- 2004 , Software Engineering Standards Committee of the IEEE Computer Society, New York, NY, 2004 .
21. B. T. Rearden, M. T. Sieger, S. M. Bowman, J. P. Lefebvre, "Quality Assurance Plan for the SCALE Code System," *SCALE-QAP-005, Rev. 4*, May 22, 2013.
22. B. T. Rearden, *SCALE Quality Assurance Program Training*, July 26, 2013.
23. B. T. Rearden, private communication, November 13, 2015.
24. B. T. Rearden, S. M. Bowman, J. P. Lefebvre, "Configuration Management Plan for the SCALE Code System," *SCALE-CMP-001, Rev. 8*, May 22, 2013.
25. U.S. Nuclear Regulatory Commission, "Potential Non-conservative Error in Modeling Geometric Regions in the Keno-V.A Criticality Code," *NRC Information Notice 2005-13*, May 17, 2005.
26. W. Wieselquist, private communication, January 5, 2016.
27. Idaho National Laboratory, *Independent Verification and Validation of SAPHIRE 8 Software Design and Interface Design*, INL/EXT-09-17069 (R1), Idaho Falls, ID, March 2010.
28. Idaho National Laboratory, SAPHIRE 8 Software Configuration Management Plan, INL/EXT -09-16696 (R1), January 2010.
29. U.S. Nuclear Regulatory Commission and Idaho National Laboratory, *Systems Analysis Programs for Hands-on Integration Reliability Evaluations (SAPHIRE) Version 8 – Volume 6 Quality Assurance*, NUREG/CR-7039, March 2011.
30. Idaho National Laboratory, *SAPHIRE 8, Software Independent Verification and Validation Plan*," INL/EXT-09-15649 (R1), Idaho Falls, ID, February 2010.
31. Idaho National Laboratory, *Independent Verification and Validation of SAPHIRE Version 8 Final Report*, INL/EXT-10-18466, Idaho Falls, ID, April 2010.
32. RELAP 3D Users Group Website (http://www4vip.inl.gov/relap5/relap5doc/relap5doept1.htm) (accessed on March 16, 2016).
33. Letter by G. Apostolakis to W. Travers, Subject: Review of the Siemens Power Corporation S-RELAP5 Code to Appendix K Small-Break Loss-of-Coolant Accident Analyses, dated February 13, 2001 (link to letter).
34. Idaho National Laboratory, *RELAP5-3D Development Software Management: Software Quality Assurance Plan*, PLN-3411, Rev. 3, August 4, 2014.

35. R. R. Schultz, *RELAP5-3D Code Manual Volume V: User's Guidelines*, INL, INEEL-EXT-98-00834, Rev. 4.2, June 2014.
36. C. Davis, *Verification Testing with Installation and Developmental Assessment Problems*, RELAP5 International Users Seminar, October 23-24, 2012 (http://www4vip.inl.gov/relap5/rius/sunvalley2012/presentations/verification-testing-davis-inl.pdf).
37. G. Mesina, *RELAP5-3D Verification 2014*, RELAP5 International Users Seminar, September 11-12, 2014 (http://www4vip.inl.gov/relap5/rius/idahofalls2014/presentations/inl-mesina-relap5-3d-verification-2014.pdf).
38. S. Bajorek, et al., "Development, Validation, and Assessment of the TRACE Thermal-Hydraulics Systems Code," *Proc. 16th Intl. Topical Mtg. Nuclear Reactor Thermal Hydraulics (NURETH-16)*, August 30-September 4, 2015.
39. N. Siu, "Software Verification and Validation: Examples from the Safety Arena," U.S. Nuclear Regulatory Commission, Office of Nuclear Regulatory Research, *INMM Workshop on VA Tools*, Boston, MA, September 14-16, 2015.
40. T. Downar, B. Kochunas, B. Collins, *MPACT Verification and Validation: Status and Plans (Rev. 1)," consortium for Advanced Simulation of LWRs Report*, CASL-X-2015-0134-000, 2015.
41. M. Sieger, *CASL-QA-030, CASL Software Quality Assurance Requirements*, January, 2015, CASL-U-2015-0010-000.

APPENDIX A. OBJECT-ORIENTED AND PROCEDURAL TEST EXAMPLES

A.1. OBJECT-ORIENTED TEST

Object oriented unit test example, ***tstConcentrationConverter.cpp*:**

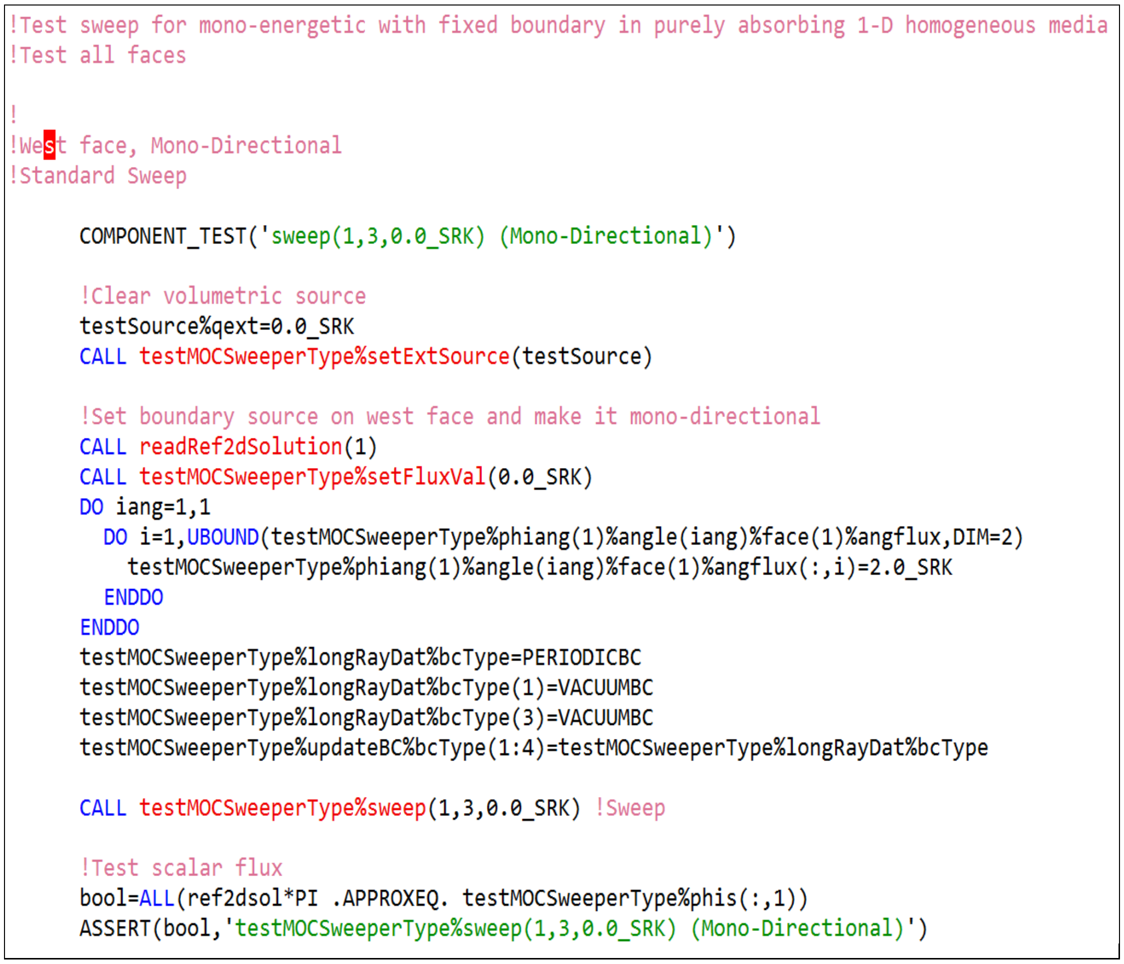
```
TEST(ConcentrationConverter,DecayUnits)
{
…
   EXPECT_TRUE(cv.convertible_to(CURIES,id,MOLES) );
   EXPECT_TRUE(cv.convertible_to(BECQUERELS,id,MOLES) );
   EXPECT_TRUE(cv.convertible_to(WATTS,id,MOLES) );
   EXPECT_TRUE(cv.convertible_to(GAMWATTS,id,MOLES) );
   EXPECT_TRUE(cv.convertible_to(MEVS,id,MOLES) );
   EXPECT_TRUE(cv.convertible_to(GAMMEVS,id,MOLES) );
   EXPECT_TRUE(cv.convertible_to(AIRM_3,id,MOLES) );
   EXPECT_TRUE(cv.convertible_to(H2OM_3,id,MOLES) );

   //reference values
   double c = 0.16674607;
   double b = 6.1696046e+09;
   double w = 0.0048892936;
   double g = 3.1766231e-05;
   double m = 3.0516558e+10;
   double v = 1.9826914e+08;
   double a = 5.0073898e+12;
   double h = 992536.12;

   //check the values from the forward
   EXPECT_FLOAT_EQ(c , cv.convert_to(CURIES,id,MOLES,1.0) );
   EXPECT_FLOAT_EQ(b , cv.convert_to(BECQUERELS,id,MOLES,1.0) );
   EXPECT_FLOAT_EQ(w , cv.convert_to(WATTS,id,MOLES,1.0) );
   EXPECT_FLOAT_EQ(g , cv.convert_to(GAMWATTS,id,MOLES,1.0) );
   EXPECT_FLOAT_EQ(m , cv.convert_to(MEVS,id,MOLES,1.0) );
   EXPECT_FLOAT_EQ(v , cv.convert_to(GAMMEVS,id,MOLES,1.0) );
   EXPECT_FLOAT_EQ(a , cv.convert_to(AIRM_3,id,MOLES,1.0) );
   EXPECT_FLOAT_EQ(h , cv.convert_to(H2OM_3,id,MOLES,1.0) );
   //now reverse
   EXPECT_FLOAT_EQ(1.0, cv.convert_to(MOLES,id,CURIES,c) );
   EXPECT_FLOAT_EQ(1.0, cv.convert_to(MOLES,id,BECQUERELS,b) );
   EXPECT_FLOAT_EQ(1.0, cv.convert_to(MOLES,id,WATTS,w) );
   EXPECT_FLOAT_EQ(1.0, cv.convert_to(MOLES,id,GAMWATTS,g) );
   EXPECT_FLOAT_EQ(1.0, cv.convert_to(MOLES,id,MEVS,m) );
   EXPECT_FLOAT_EQ(1.0, cv.convert_to(MOLES,id,GAMMEVS,v) );
   EXPECT_FLOAT_EQ(1.0, cv.convert_to(MOLES,id,AIRM_3,a) );
   EXPECT_FLOAT_EQ(1.0, cv.convert_to(MOLES,id,H2OM_3,h) );
}
```

A.2. PROCEDURAL TEST

Procedural programming unit test example, *tstNuclideSet.f90*:

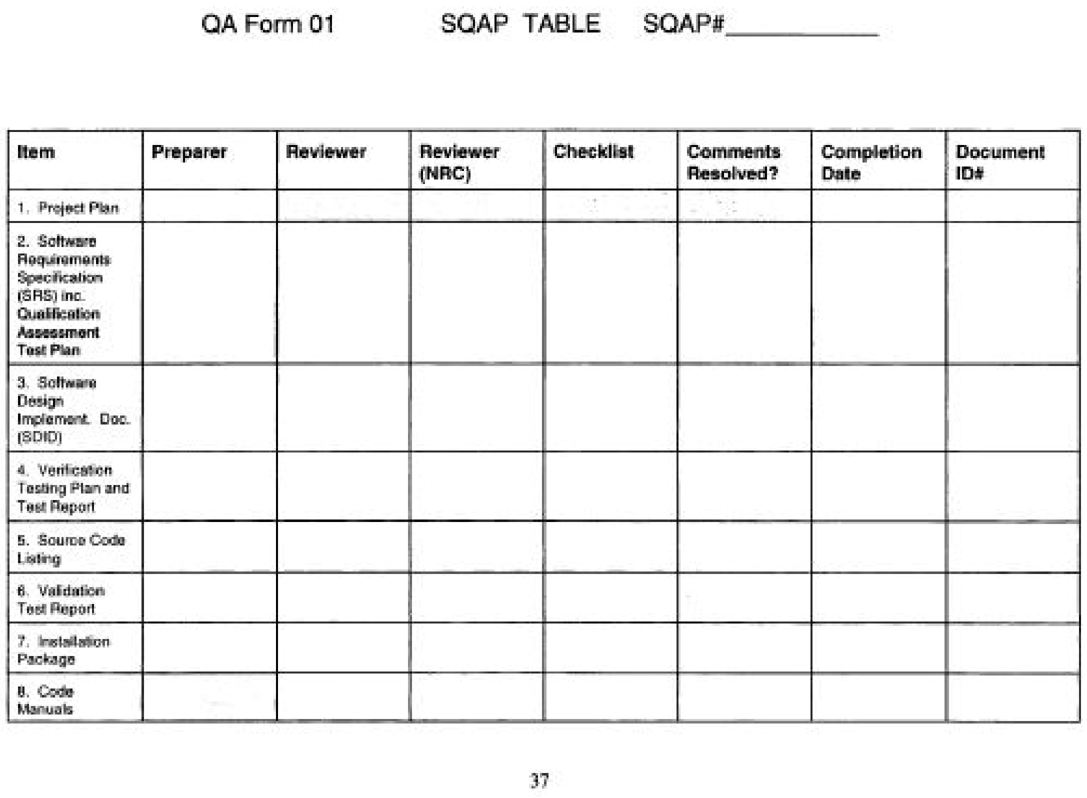
```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Test symbol_to_izzzaaa
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
call vec_ids % initialize()
call vec_str % initialize()
call vec_str % push_back("u235")
call vec_str % push_back("u235m")
call vec_str % push_back("pu239")
call vec_str % push_back("h1")
call vec_str % push_back("zr")
call nucset % convert_symbol_to_izzzaaa(vec_str,vec_ids)
EXPECT_EQ(92235,vec_ids%at(1_L))
EXPECT_EQ(1092235,vec_ids%at(2_L))
EXPECT_EQ(94239,vec_ids%at(3_L))
EXPECT_EQ(1001,vec_ids%at(4_L))
EXPECT_EQ(40000,vec_ids%at(5_L))

call nucset % convert_sizzzaaa_to_zzzaaai(vec_ids)
EXPECT_EQ(922350,vec_ids%at(1_L))
EXPECT_EQ(922351,vec_ids%at(2_L))
EXPECT_EQ(942390,vec_ids%at(3_L))
EXPECT_EQ( 10010,vec_ids%at(4_L))
EXPECT_EQ(400000,vec_ids%at(5_L))
```

APPENDIX B. CODE SNIPPET FROM MPACT UNIT TEST



```
!Test sweep for mono-energetic with fixed boundary in purely absorbing 1-D homogeneous media
!Test all faces

!
!West face, Mono-Directional
!Standard Sweep

      COMPONENT_TEST('sweep(1,3,0.0_SRK) (Mono-Directional)')

      !Clear volumetric source
      testSource%qext=0.0_SRK
      CALL testMOCSweeperType%setExtSource(testSource)

      !Set boundary source on west face and make it mono-directional
      CALL readRef2dSolution(1)
      CALL testMOCSweeperType%setFluxVal(0.0_SRK)
      DO iang=1,1
        DO i=1,UBOUND(testMOCSweeperType%phiang(1)%angle(iang)%face(1)%angflux,DIM=2)
          testMOCSweeperType%phiang(1)%angle(iang)%face(1)%angflux(:,i)=2.0_SRK
        ENDDO
      ENDDO
      testMOCSweeperType%longRayDat%bcType=PERIODICBC
      testMOCSweeperType%longRayDat%bcType(1)=VACUUMBC
      testMOCSweeperType%longRayDat%bcType(3)=VACUUMBC
      testMOCSweeperType%updateBC%bcType(1:4)=testMOCSweeperType%longRayDat%bcType

      CALL testMOCSweeperType%sweep(1,3,0.0_SRK) !Sweep

      !Test scalar flux
      bool=ALL(ref2dsol*PI .APPROXEQ. testMOCSweeperType%phis(:,1))
      ASSERT(bool,'testMOCSweeperType%sweep(1,3,0.0_SRK) (Mono-Directional)')
```

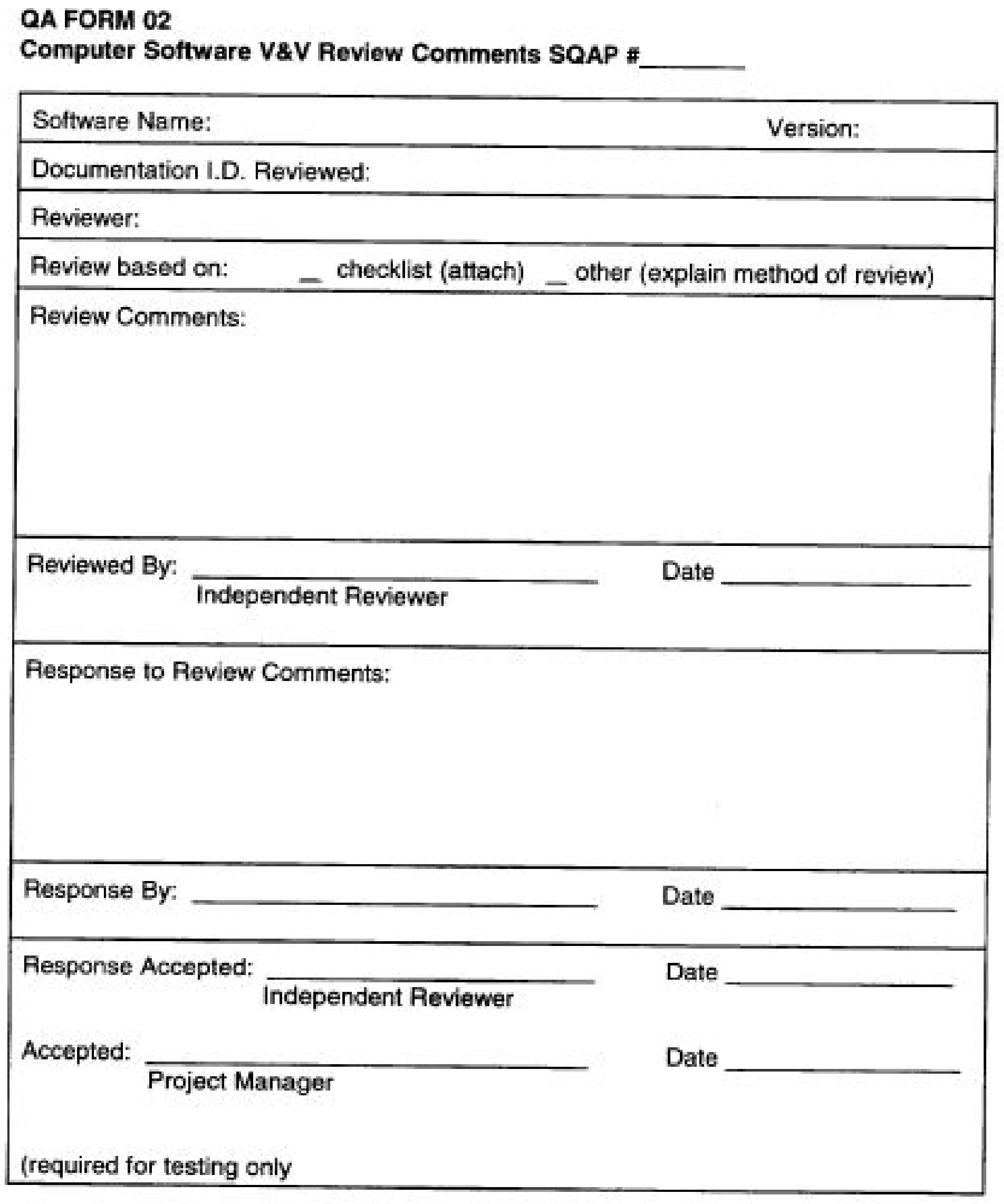
APPENDIX C. EXAMPLE CHECKLISTS FROM NUREG-1737 A



QA Form 01 SQAP TABLE SQAP#__________

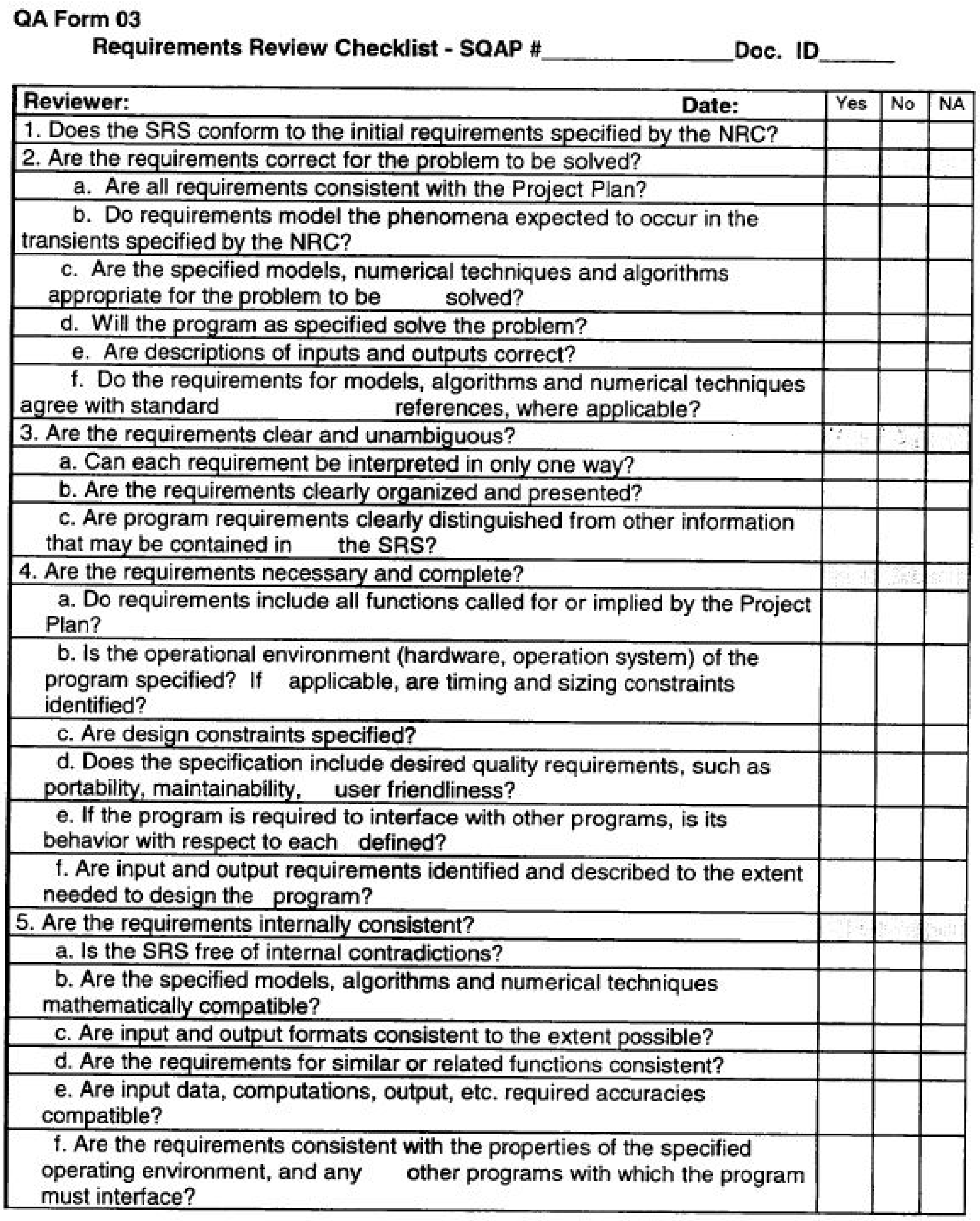
Item	Preparer	Reviewer	Reviewer (NRC)	Checklist	Comments Resolved?	Completion Date	Document ID#
1. Project Plan							
2. Software Requirements Specification (SRS) inc. Qualification Assessment Test Plan							
3. Software Design Implement. Doc. (SDID)							
4. Verification Testing Plan and Test Report							
5. Source Code Listing							
6. Validation Test Report							
7. Installation Package							
8. Code Manuals							

37



QA FORM 02
Computer Software V&V Review Comments SQAP #________

Software Name:	Version:
Documentation I.D. Reviewed:	
Reviewer:	
Review based on: _ checklist (attach) _ other (explain method of review)	
Review Comments:	
Reviewed By: ______________________ Independent Reviewer	Date ____________
Response to Review Comments:	
Response By: ______________________	Date ____________
Response Accepted: ______________________ Independent Reviewer Accepted: ______________________ Project Manager (required for testing only	Date ____________ Date ____________

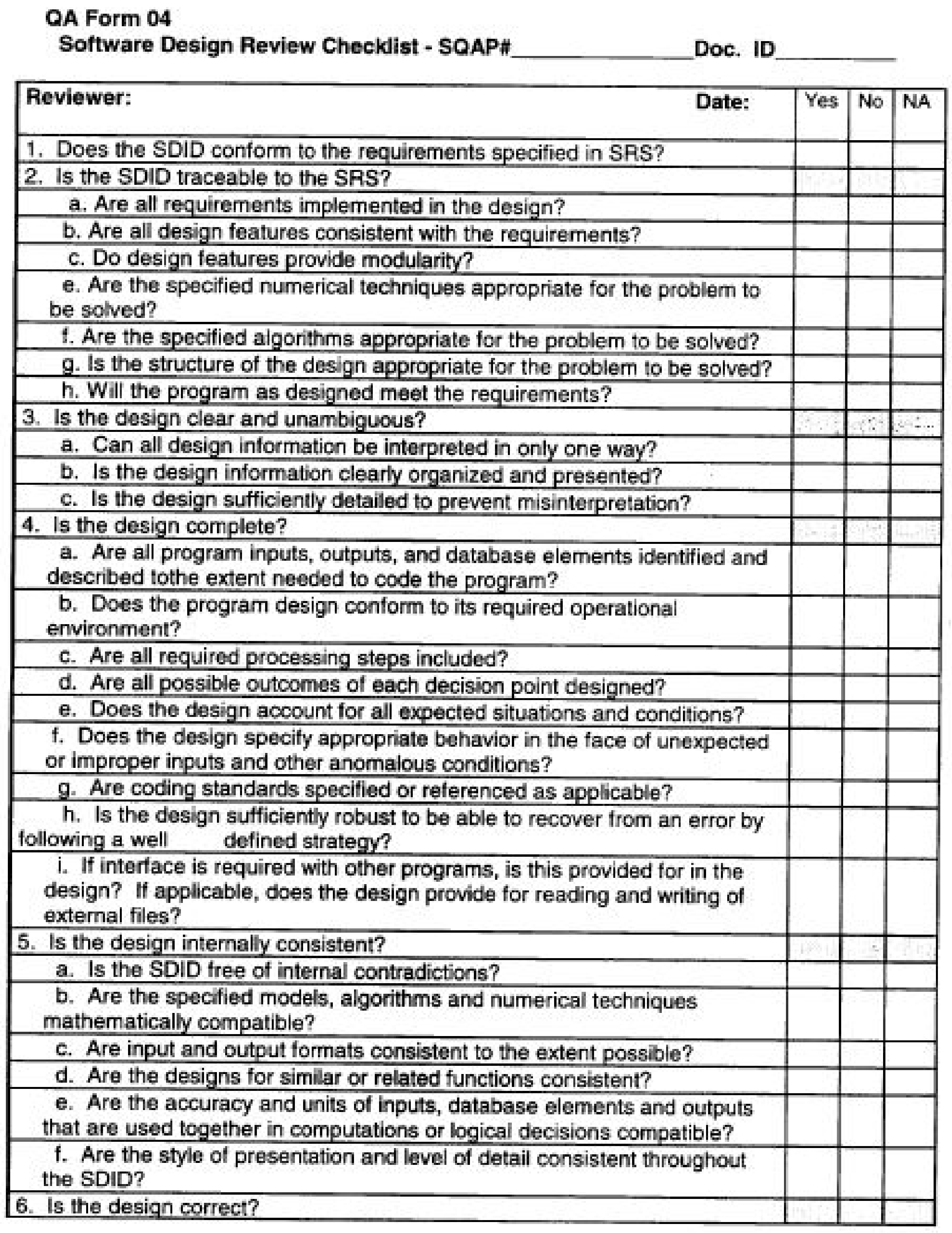


QA Form 03

Requirements Review Checklist - SQAP #______________Doc. ID______

Reviewer: **Date:**	Yes	No	NA
1. Does the SRS conform to the initial requirements specified by the NRC?			
2. Are the requirements correct for the problem to be solved?			
a. Are all requirements consistent with the Project Plan?			
b. Do requirements model the phenomena expected to occur in the transients specified by the NRC?			
c. Are the specified models, numerical techniques and algorithms appropriate for the problem to be solved?			
d. Will the program as specified solve the problem?			
e. Are descriptions of inputs and outputs correct?			
f. Do the requirements for models, algorithms and numerical techniques agree with standard references, where applicable?			
3. Are the requirements clear and unambiguous?			
a. Can each requirement be interpreted in only one way?			
b. Are the requirements clearly organized and presented?			
c. Are program requirements clearly distinguished from other information that may be contained in the SRS?			
4. Are the requirements necessary and complete?			
a. Do requirements include all functions called for or implied by the Project Plan?			
b. Is the operational environment (hardware, operation system) of the program specified? If applicable, are timing and sizing constraints identified?			
c. Are design constraints specified?			
d. Does the specification include desired quality requirements, such as portability, maintainability, user friendliness?			
e. If the program is required to interface with other programs, is its behavior with respect to each defined?			
f. Are input and output requirements identified and described to the extent needed to design the program?			
5. Are the requirements internally consistent?			
a. Is the SRS free of internal contradictions?			
b. Are the specified models, algorithms and numerical techniques mathematically compatible?			
c. Are input and output formats consistent to the extent possible?			
d. Are the requirements for similar or related functions consistent?			
e. Are input data, computations, output, etc. required accuracies compatible?			
f. Are the requirements consistent with the properties of the specified operating environment, and any other programs with which the program must interface?			

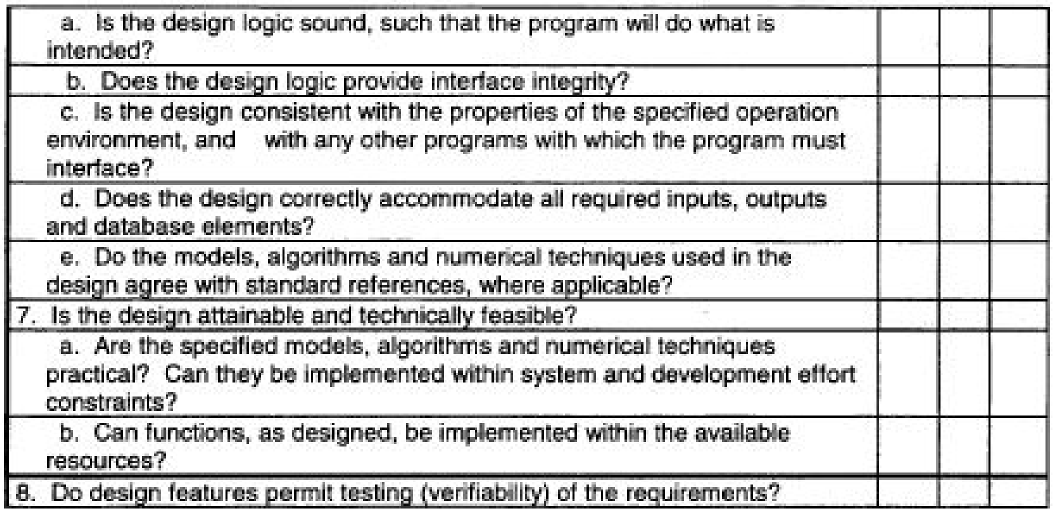
6. Are the requirements feasible?			
a. Are the specified models, algorithms and numerical techniques practical? Can they be implemented within system and development effort constraints?			
b. Are the required functions attainable within the available resources?			
c. Can the desired attributes be achieved individually and as a group? (ie, generally not possible to maximize both efficiency and maintainability.)			
7. Do the requirements make adequate provision for program V&V testing?			
a. Is each requirement testable?			
b. Are acceptance criteria specified?			
c. Are the acceptance criteria consistent with at least one of the following (circle all appropriate): Results obtained from similar computer programs; Solutions of classical problems; Accepted experimental results; Analytical results published in technical literature; Solutions of benchmark problems			



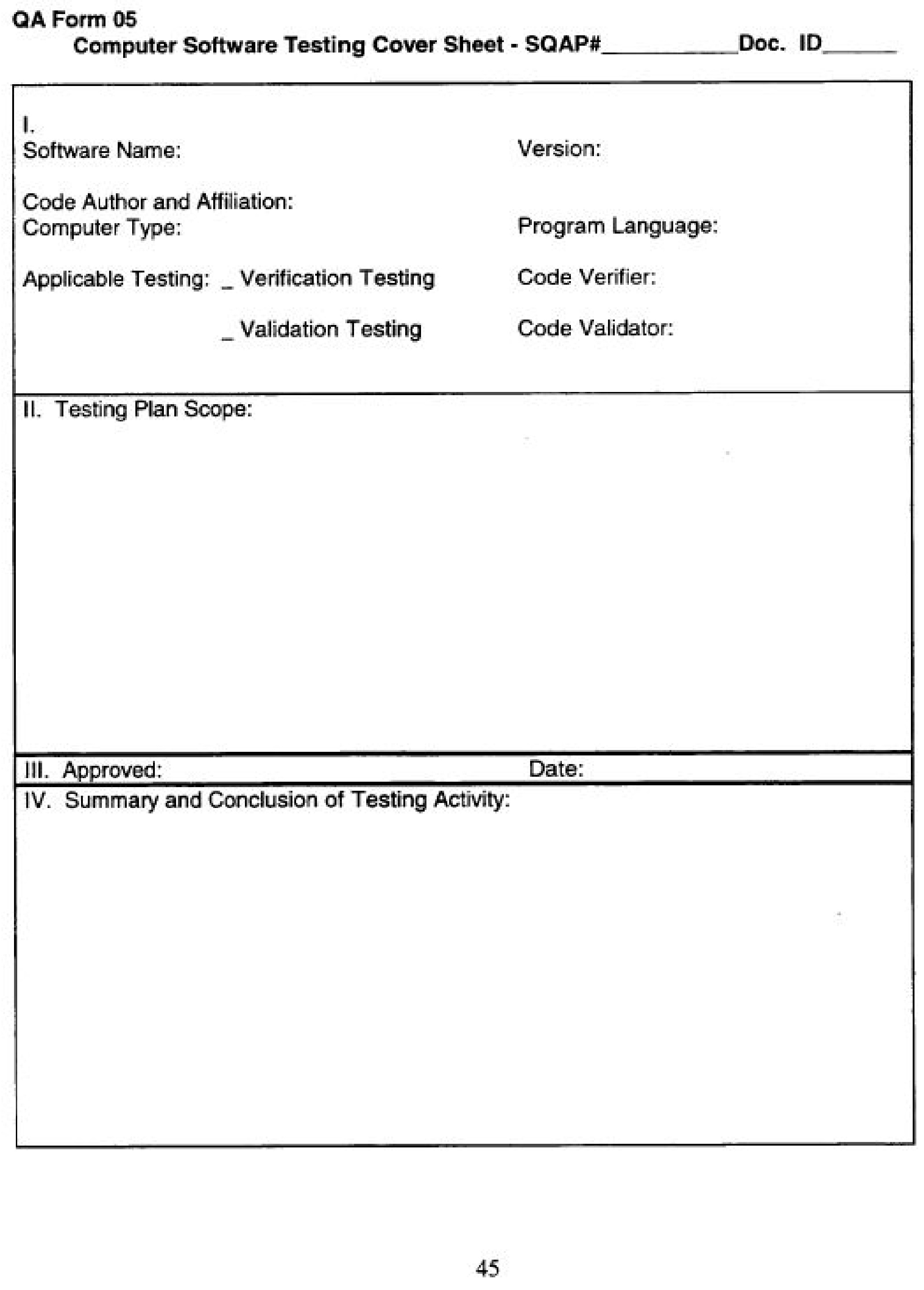
QA Form 04

Software Design Review Checklist - SQAP#______________Doc. ID__________

Reviewer: **Date:**	Yes	No	NA
1. Does the SDID conform to the requirements specified in SRS?			
2. Is the SDID traceable to the SRS?			
a. Are all requirements implemented in the design?			
b. Are all design features consistent with the requirements?			
c. Do design features provide modularity?			
e. Are the specified numerical techniques appropriate for the problem to be solved?			
f. Are the specified algorithms appropriate for the problem to be solved?			
g. Is the structure of the design appropriate for the problem to be solved?			
h. Will the program as designed meet the requirements?			
3. Is the design clear and unambiguous?			
a. Can all design information be interpreted in only one way?			
b. Is the design information clearly organized and presented?			
c. Is the design sufficiently detailed to prevent misinterpretation?			
4. Is the design complete?			
a. Are all program inputs, outputs, and database elements identified and described tothe extent needed to code the program?			
b. Does the program design conform to its required operational environment?			
c. Are all required processing steps included?			
d. Are all possible outcomes of each decision point designed?			
e. Does the design account for all expected situations and conditions?			
f. Does the design specify appropriate behavior in the face of unexpected or improper inputs and other anomalous conditions?			
g. Are coding standards specified or referenced as applicable?			
h. Is the design sufficiently robust to be able to recover from an error by following a well defined strategy?			
i. If interface is required with other programs, is this provided for in the design? If applicable, does the design provide for reading and writing of external files?			
5. Is the design internally consistent?			
a. Is the SDID free of internal contradictions?			
b. Are the specified models, algorithms and numerical techniques mathematically compatible?			
c. Are input and output formats consistent to the extent possible?			
d. Are the designs for similar or related functions consistent?			
e. Are the accuracy and units of inputs, database elements and outputs that are used together in computations or logical decisions compatible?			
f. Are the style of presentation and level of detail consistent throughout the SDID?			
6. Is the design correct?			



a. Is the design logic sound, such that the program will do what is intended?			
b. Does the design logic provide interface integrity?			
c. Is the design consistent with the properties of the specified operation environment, and with any other programs with which the program must interface?			
d. Does the design correctly accommodate all required inputs, outputs and database elements?			
e. Do the models, algorithms and numerical techniques used in the design agree with standard references, where applicable?			
7. Is the design attainable and technically feasible?			
a. Are the specified models, algorithms and numerical techniques practical? Can they be implemented within system and development effort constraints?			
b. Can functions, as designed, be implemented within the available resources?			
8. Do design features permit testing (verifiability) of the requirements?			

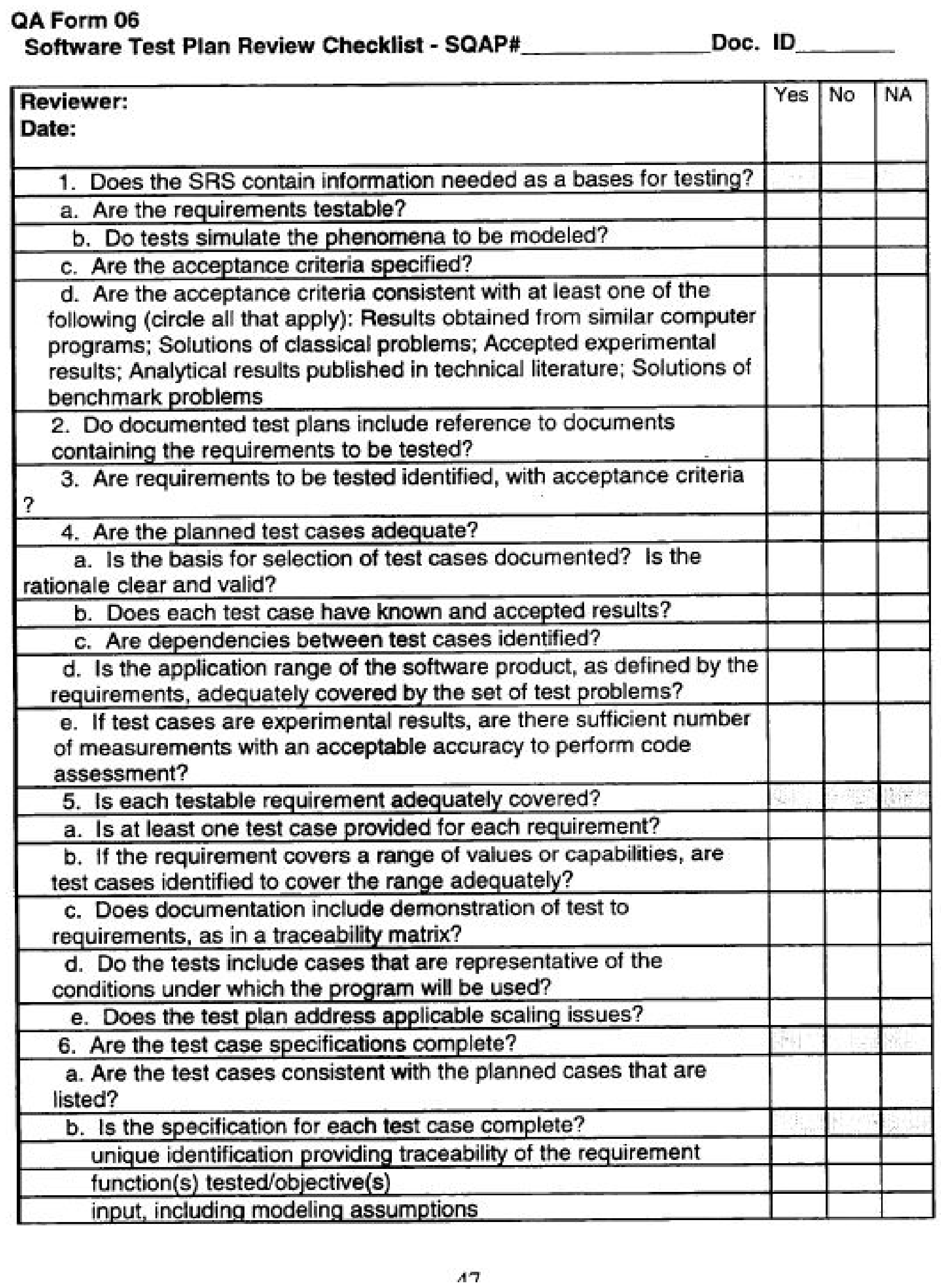


QA Form 05

Computer Software Testing Cover Sheet - SQAP#__________Doc. ID______

I. Software Name:	Version:
Code Author and Affiliation: Computer Type:	Program Language:
Applicable Testing: _ Verification Testing	Code Verifier:
_ Validation Testing	Code Validator:
II. Testing Plan Scope:	
III. Approved:	Date:
IV. Summary and Conclusion of Testing Activity:	

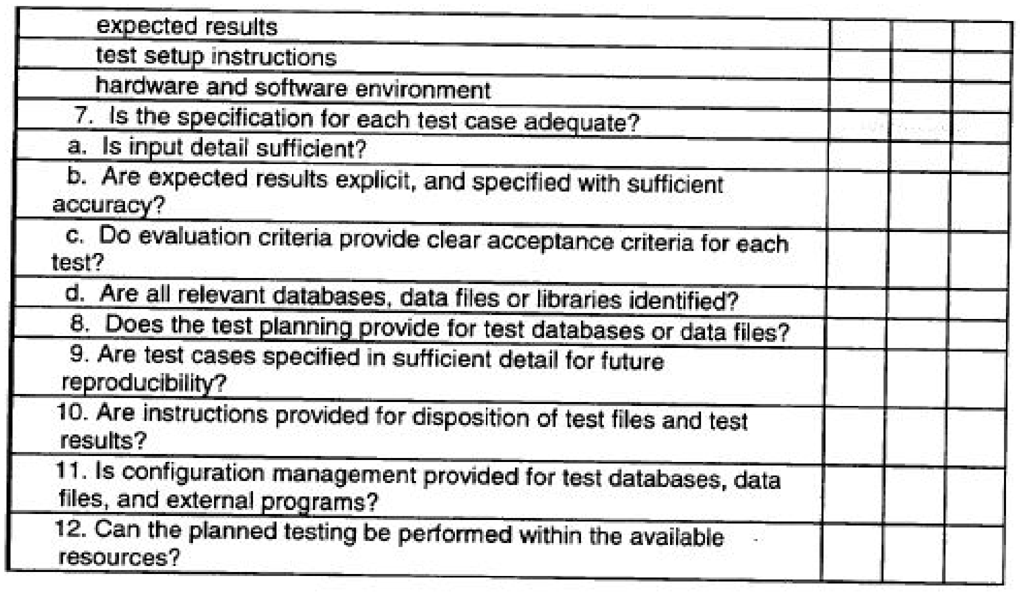
45



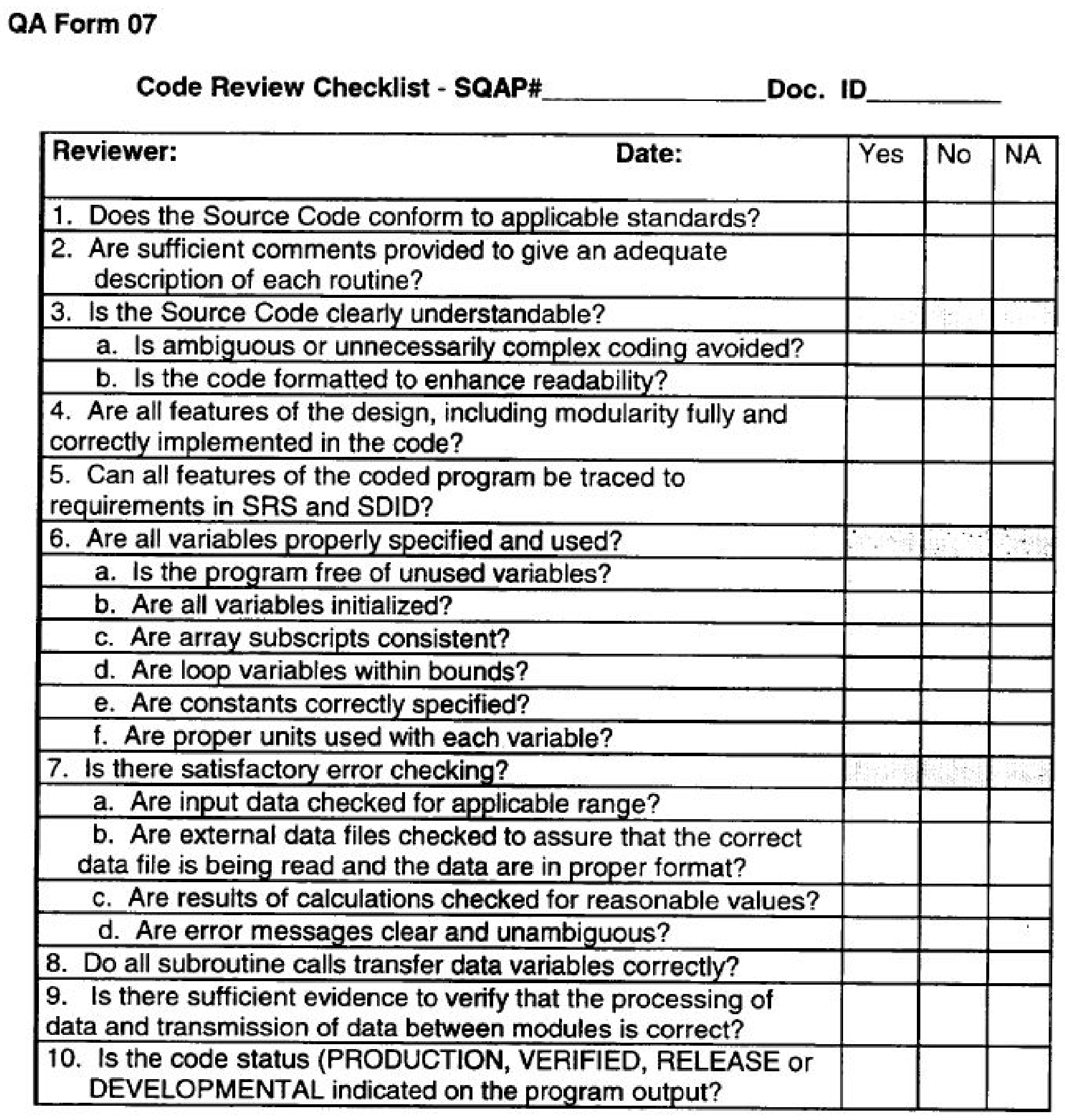
QA Form 06
Software Test Plan Review Checklist - SQAP#_______________Doc. ID________

Reviewer: **Date:**	Yes	No	NA
1. Does the SRS contain information needed as a bases for testing?			
a. Are the requirements testable?			
b. Do tests simulate the phenomena to be modeled?			
c. Are the acceptance criteria specified?			
d. Are the acceptance criteria consistent with at least one of the following (circle all that apply): Results obtained from similar computer programs; Solutions of classical problems; Accepted experimental results; Analytical results published in technical literature; Solutions of benchmark problems			
2. Do documented test plans include reference to documents containing the requirements to be tested?			
3. Are requirements to be tested identified, with acceptance criteria ?			
4. Are the planned test cases adequate?			
a. Is the basis for selection of test cases documented? Is the rationale clear and valid?			
b. Does each test case have known and accepted results?			
c. Are dependencies between test cases identified?			
d. Is the application range of the software product, as defined by the requirements, adequately covered by the set of test problems?			
e. If test cases are experimental results, are there sufficient number of measurements with an acceptable accuracy to perform code assessment?			
5. Is each testable requirement adequately covered?			
a. Is at least one test case provided for each requirement?			
b. If the requirement covers a range of values or capabilities, are test cases identified to cover the range adequately?			
c. Does documentation include demonstration of test to requirements, as in a traceability matrix?			
d. Do the tests include cases that are representative of the conditions under which the program will be used?			
e. Does the test plan address applicable scaling issues?			
6. Are the test case specifications complete?			
a. Are the test cases consistent with the planned cases that are listed?			
b. Is the specification for each test case complete?			
unique identification providing traceability of the requirement			
function(s) tested/objective(s)			
input, including modeling assumptions			

47



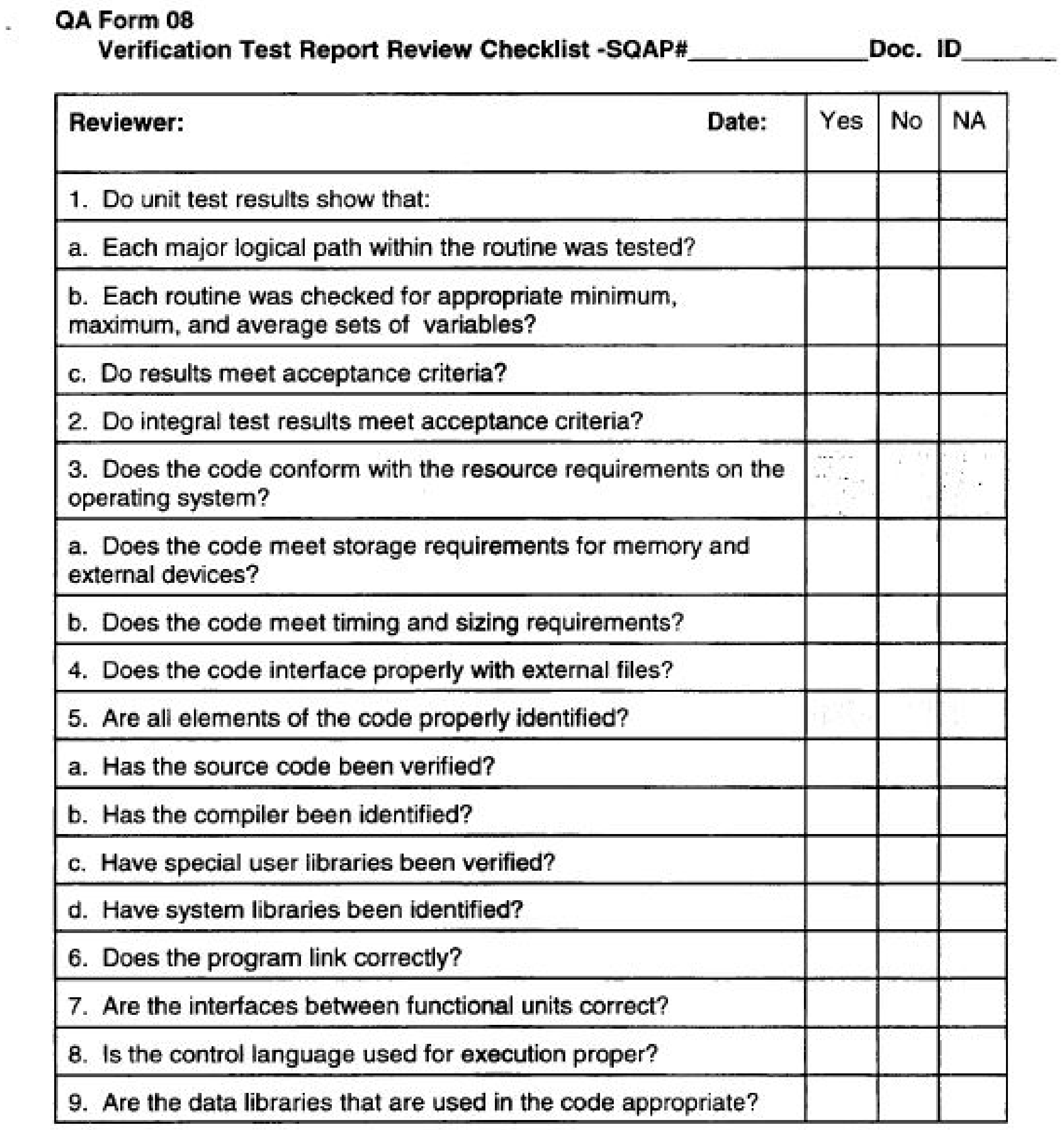
expected results			
test setup instructions			
hardware and software environment			
7. Is the specification for each test case adequate?			
a. Is input detail sufficient?			
b. Are expected results explicit, and specified with sufficient accuracy?			
c. Do evaluation criteria provide clear acceptance criteria for each test?			
d. Are all relevant databases, data files or libraries identified?			
8. Does the test planning provide for test databases or data files?			
9. Are test cases specified in sufficient detail for future reproducibility?			
10. Are instructions provided for disposition of test files and test results?			
11. Is configuration management provided for test databases, data files, and external programs?			
12. Can the planned testing be performed within the available resources?			



QA Form 07

Code Review Checklist - SQAP#______________Doc. ID_________

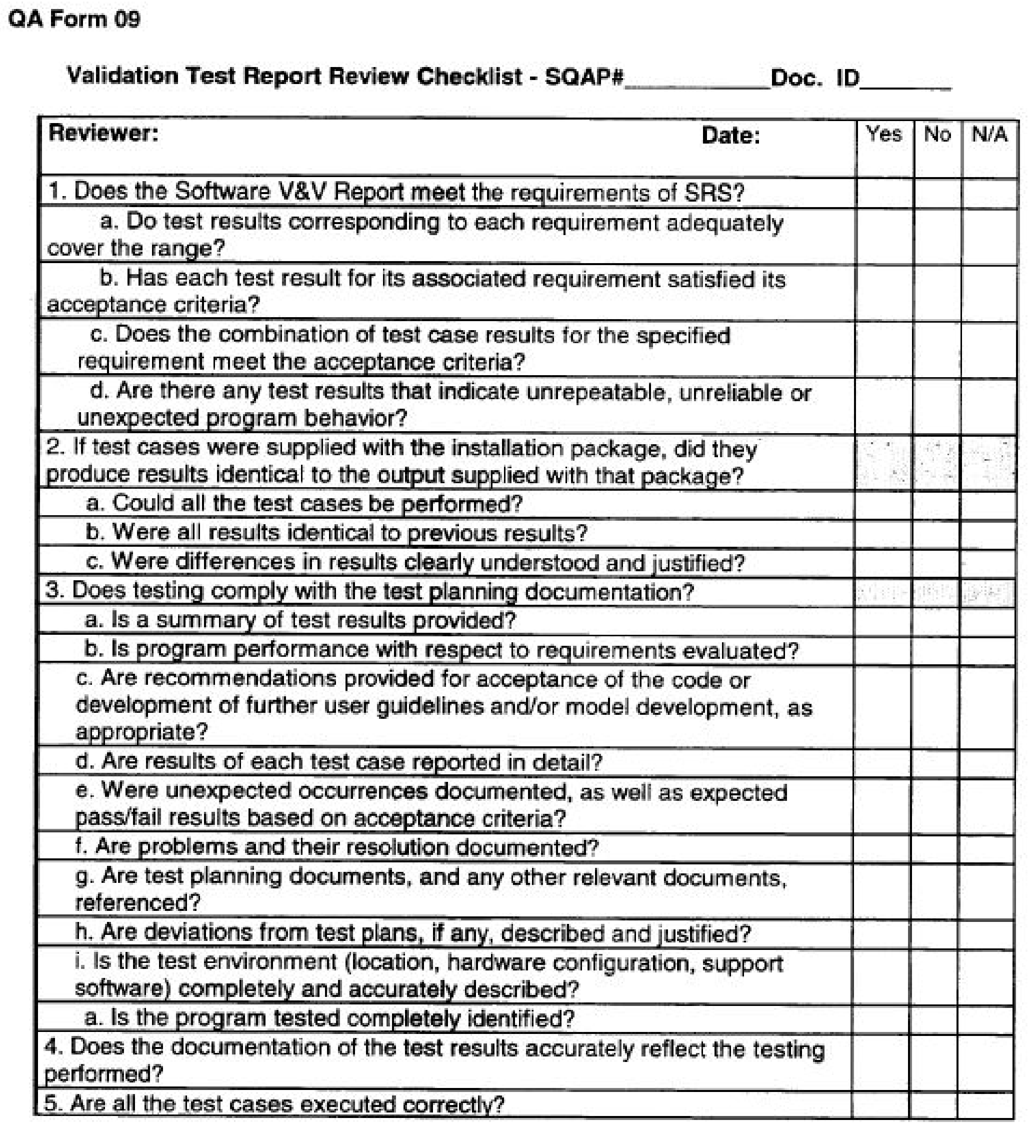
Reviewer: **Date:**	Yes	No	NA
1. Does the Source Code conform to applicable standards?			
2. Are sufficient comments provided to give an adequate description of each routine?			
3. Is the Source Code clearly understandable?			
a. Is ambiguous or unnecessarily complex coding avoided?			
b. Is the code formatted to enhance readability?			
4. Are all features of the design, including modularity fully and correctly implemented in the code?			
5. Can all features of the coded program be traced to requirements in SRS and SDID?			
6. Are all variables properly specified and used?			
a. Is the program free of unused variables?			
b. Are all variables initialized?			
c. Are array subscripts consistent?			
d. Are loop variables within bounds?			
e. Are constants correctly specified?			
f. Are proper units used with each variable?			
7. Is there satisfactory error checking?			
a. Are input data checked for applicable range?			
b. Are external data files checked to assure that the correct data file is being read and the data are in proper format?			
c. Are results of calculations checked for reasonable values?			
d. Are error messages clear and unambiguous?			
8. Do all subroutine calls transfer data variables correctly?			
9. Is there sufficient evidence to verify that the processing of data and transmission of data between modules is correct?			
10. Is the code status (PRODUCTION, VERIFIED, RELEASE or DEVELOPMENTAL indicated on the program output?			



QA Form 08

Verification Test Report Review Checklist -SQAP#____________Doc. ID______

Reviewer: **Date:**	Yes	No	NA
1. Do unit test results show that:			
a. Each major logical path within the routine was tested?			
b. Each routine was checked for appropriate minimum, maximum, and average sets of variables?			
c. Do results meet acceptance criteria?			
2. Do integral test results meet acceptance criteria?			
3. Does the code conform with the resource requirements on the operating system?			
a. Does the code meet storage requirements for memory and external devices?			
b. Does the code meet timing and sizing requirements?			
4. Does the code interface properly with external files?			
5. Are all elements of the code properly identified?			
a. Has the source code been verified?			
b. Has the compiler been identified?			
c. Have special user libraries been verified?			
d. Have system libraries been identified?			
6. Does the program link correctly?			
7. Are the interfaces between functional units correct?			
8. Is the control language used for execution proper?			
9. Are the data libraries that are used in the code appropriate?			

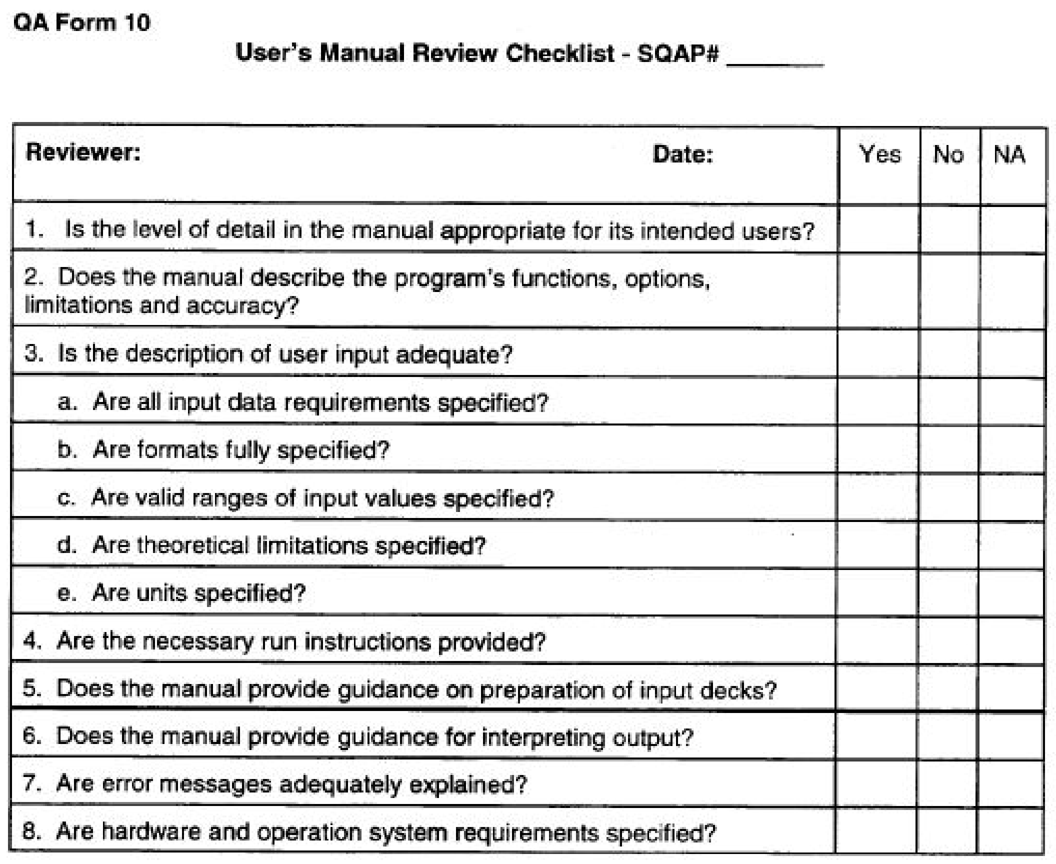


QA Form 09

Validation Test Report Review Checklist - SQAP#___________Doc. ID_______

Reviewer: **Date:**	Yes	No	N/A
1. Does the Software V&V Report meet the requirements of SRS?			
a. Do test results corresponding to each requirement adequately cover the range?			
b. Has each test result for its associated requirement satisfied its acceptance criteria?			
c. Does the combination of test case results for the specified requirement meet the acceptance criteria?			
d. Are there any test results that indicate unrepeatable, unreliable or unexpected program behavior?			
2. If test cases were supplied with the installation package, did they produce results identical to the output supplied with that package?	▨	▨	▨
a. Could all the test cases be performed?			
b. Were all results identical to previous results?			
c. Were differences in results clearly understood and justified?			
3. Does testing comply with the test planning documentation?	▨	▨	▨
a. Is a summary of test results provided?			
b. Is program performance with respect to requirements evaluated?			
c. Are recommendations provided for acceptance of the code or development of further user guidelines and/or model development, as appropriate?			
d. Are results of each test case reported in detail?			
e. Were unexpected occurrences documented, as well as expected pass/fail results based on acceptance criteria?			
f. Are problems and their resolution documented?			
g. Are test planning documents, and any other relevant documents, referenced?			
h. Are deviations from test plans, if any, described and justified?			
i. Is the test environment (location, hardware configuration, support software) completely and accurately described?			
a. Is the program tested completely identified?			
4. Does the documentation of the test results accurately reflect the testing performed?			
5. Are all the test cases executed correctly?			

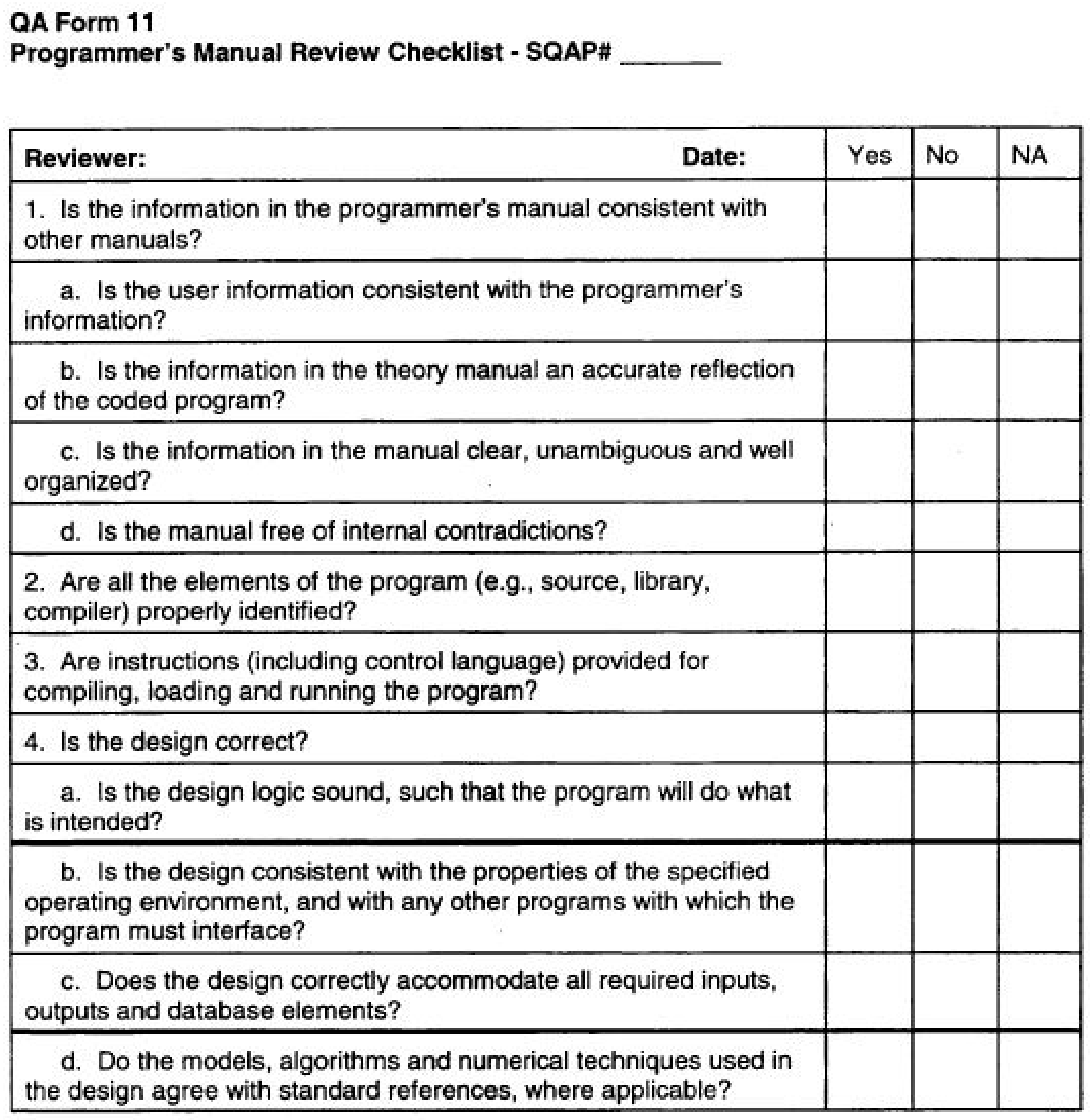
a. Do test results adequately identify the software and hardware under test, including support software such as operation system, test drivers, test data?			
b. Do reported test results indicate performance of each test case in the specified environment, using the documented specifications?			
c. Is there an explanation for any deviation from the specified test environment or procedures?			
d. Is there a problem report for each deviation from expected results?			
e. Were correct input data used for each test case?			
f. Is the output of each test case accurately reported or attached?			



QA Form 10

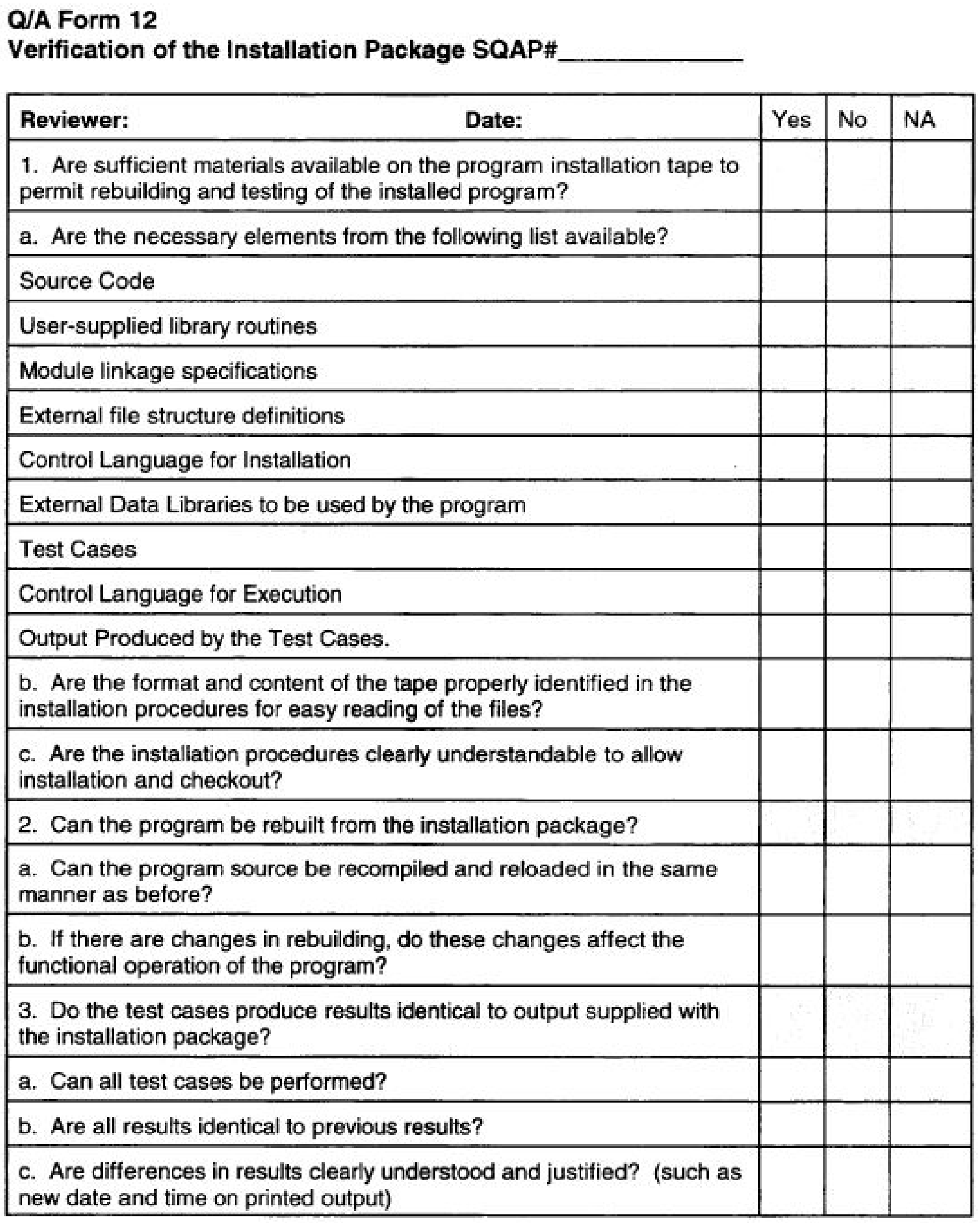
User's Manual Review Checklist - SQAP# _______

Reviewer: **Date:**	Yes	No	NA
1. Is the level of detail in the manual appropriate for its intended users?			
2. Does the manual describe the program's functions, options, limitations and accuracy?			
3. Is the description of user input adequate?			
a. Are all input data requirements specified?			
b. Are formats fully specified?			
c. Are valid ranges of input values specified?			
d. Are theoretical limitations specified?			
e. Are units specified?			
4. Are the necessary run instructions provided?			
5. Does the manual provide guidance on preparation of input decks?			
6. Does the manual provide guidance for interpreting output?			
7. Are error messages adequately explained?			
8. Are hardware and operation system requirements specified?			



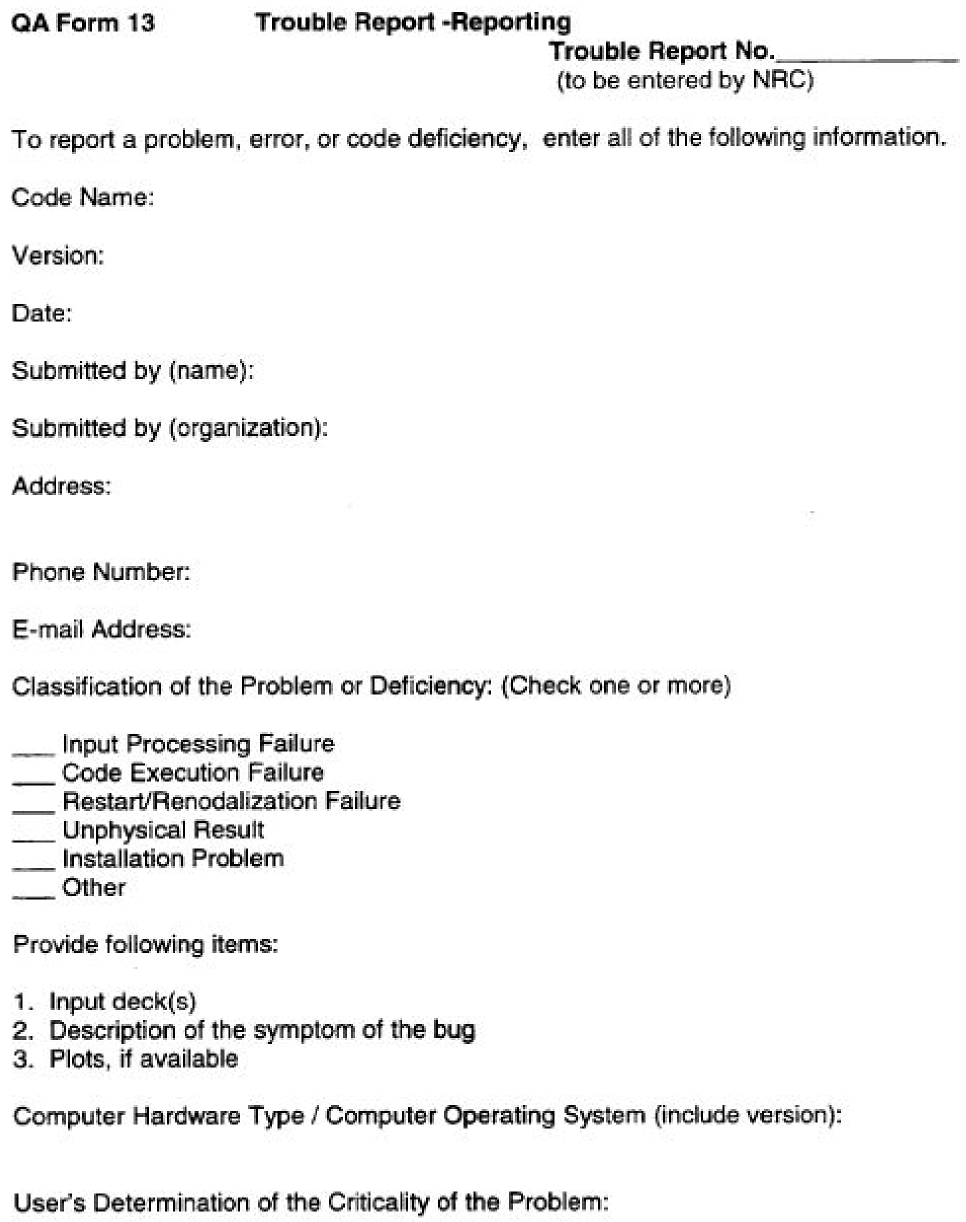
QA Form 11
Programmer's Manual Review Checklist - SQAP# _______

Reviewer: **Date:**	Yes	No	NA
1. Is the information in the programmer's manual consistent with other manuals?			
a. Is the user information consistent with the programmer's information?			
b. Is the information in the theory manual an accurate reflection of the coded program?			
c. Is the information in the manual clear, unambiguous and well organized?			
d. Is the manual free of internal contradictions?			
2. Are all the elements of the program (e.g., source, library, compiler) properly identified?			
3. Are instructions (including control language) provided for compiling, loading and running the program?			
4. Is the design correct?			
a. Is the design logic sound, such that the program will do what is intended?			
b. Is the design consistent with the properties of the specified operating environment, and with any other programs with which the program must interface?			
c. Does the design correctly accommodate all required inputs, outputs and database elements?			
d. Do the models, algorithms and numerical techniques used in the design agree with standard references, where applicable?			



Q/A Form 12
Verification of the Installation Package SQAP#_____________

Reviewer: **Date:**	Yes	No	NA
1. Are sufficient materials available on the program installation tape to permit rebuilding and testing of the installed program?			
a. Are the necessary elements from the following list available?			
Source Code			
User-supplied library routines			
Module linkage specifications			
External file structure definitions			
Control Language for Installation			
External Data Libraries to be used by the program			
Test Cases			
Control Language for Execution			
Output Produced by the Test Cases.			
b. Are the format and content of the tape properly identified in the installation procedures for easy reading of the files?			
c. Are the installation procedures clearly understandable to allow installation and checkout?			
2. Can the program be rebuilt from the installation package?			
a. Can the program source be recompiled and reloaded in the same manner as before?			
b. If there are changes in rebuilding, do these changes affect the functional operation of the program?			
3. Do the test cases produce results identical to output supplied with the installation package?			
a. Can all test cases be performed?			
b. Are all results identical to previous results?			
c. Are differences in results clearly understood and justified? (such as new date and time on printed output)			



QA Form 13 **Trouble Report -Reporting**

Trouble Report No.______________
(to be entered by NRC)

To report a problem, error, or code deficiency, enter all of the following information.

Code Name:

Version:

Date:

Submitted by (name):

Submitted by (organization):

Address:

Phone Number:

E-mail Address:

Classification of the Problem or Deficiency: (Check one or more)

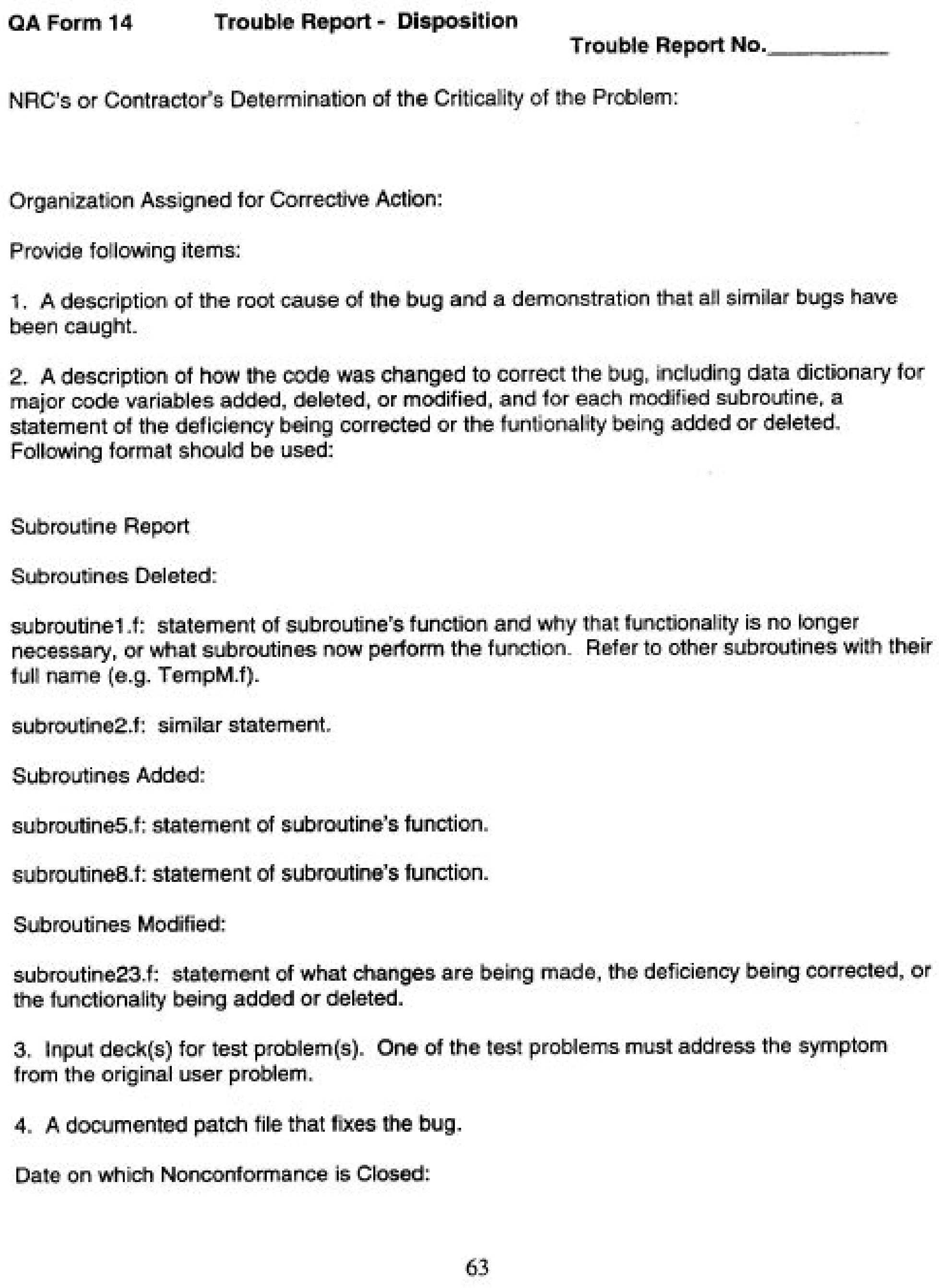
___ Input Processing Failure
___ Code Execution Failure
___ Restart/Renodalization Failure
___ Unphysical Result
___ Installation Problem
___ Other

Provide following items:

1. Input deck(s)
2. Description of the symptom of the bug
3. Plots, if available

Computer Hardware Type / Computer Operating System (include version):

User's Determination of the Criticality of the Problem:



QA Form 14 **Trouble Report - Disposition**

Trouble Report No.__________

NRC's or Contractor's Determination of the Criticality of the Problem:

Organization Assigned for Corrective Action:

Provide following items:

1. A description of the root cause of the bug and a demonstration that all similar bugs have been caught.

2. A description of how the code was changed to correct the bug, including data dictionary for major code variables added, deleted, or modified, and for each modified subroutine, a statement of the deficiency being corrected or the funtionality being added or deleted. Following format should be used:

Subroutine Report

Subroutines Deleted:

subroutine1.f: statement of subroutine's function and why that functionality is no longer necessary, or what subroutines now perform the function. Refer to other subroutines with their full name (e.g. TempM.f).

subroutine2.f: similar statement.

Subroutines Added:

subroutine5.f: statement of subroutine's function.

subroutine8.f: statement of subroutine's function.

Subroutines Modified:

subroutine23.f: statement of what changes are being made, the deficiency being corrected, or the functionality being added or deleted.

3. Input deck(s) for test problem(s). One of the test problems must address the symptom from the original user problem.

4. A documented patch file that fixes the bug.

Date on which Nonconformance is Closed:

63