

DOE SUBCOMMITTEE ON CONSEQUENCE ASSESSMENT AND PROTECTIVE ACTIONS
EMERGENCY MANAGEMENT ISSUES SPECIAL INTEREST GROUP

Software Quality Assurance Guidance for Consequence Assessment Software Designed for Safety-Related and Other Non-Safety Applications

C.S. Glantz and J.P. Rishel¹

SCAPA Consequence Assessment Modeling Working Group

07/30/2010

¹Pacific Northwest National Laboratory
Fundamental and Computer Sciences Directorate
902 Battelle Blvd.
Richland, WA 99352

The U.S. Department of Energy Subcommittee on Consequence Assessment and Protective Actions (SCAPA) has developed software quality assurance guidance for consequence assessment software to support applications that do not involve a direct nuclear safety function as outlined in DOE Order 414-1C, but instead provide a safety-related or other non-safety function. This guidance may be applicable to other software products that support emergency preparedness and response applications.

EXECUTIVE SUMMARY

Consequence assessment software is used to support a variety of emergency preparedness applications in the U.S. Department of Energy (DOE) complex. Some consequence assessment software products fit within the definition of safety software as defined in DOE Order 414.1C, and some do not. To provide software quality assurance (SQA) guidance for consequence assessment models and related software that are not covered by DOE O 414.1C, the Subcommittee on Consequence Assessment and Protective Actions (SCAPA) has prepared this SQA guidance document. The SCAPA guidance incorporates the key elements found in the DOE guidance for safety software but does so using an appropriately graded approach that is readily implementable by DOE's emergency management community and its software suppliers. This approach strikes an acceptable balance between the need for modeling complex environmental processes (e.g., atmospheric dispersion and deposition), timely innovation, and SQA for software products that are not classified as safety software.

The SCAPA SQA guidance is based on the SQA framework for safety software described in DOE Guide 414.1-4. For the development of new or the upgrading of existing safety-related and other non-safety consequence assessment software, SCAPA recommends minimum compliance levels for each of the ten SQA work activities described in DOE Guide 414.1-4. Many applications are likely to require additional SQA beyond the minimum levels prescribed by SCAPA to meet any applicable DOE, site, contractor, and programmatic guidance. For these applications, SCAPA presents guidance for prioritizing SQA work activities so that limited SQA resources can be allocated to activities that will provide the greatest cost benefit. Additional SQA should in general be performed to first address higher-priority work activities before attempting to meet those with medium and low priorities. The ten SQA work activities and their relative priority levels are

1. Software Project Management and Quality Planning (*High Priority*)
2. Software Risk Management (*Medium Priority*)
3. Software Configuration Management (*High Priority*)
4. Procurement and Supplier Management (*Medium Priority*)
5. Software Requirements Identification and Management (*Medium Priority*)
6. Software Design and Implementation (*High Priority*)
7. Software Safety (*Low Priority*)
8. Verification and Validation (*High Priority*)
9. Problem Reporting and Corrective Action (*High Priority*)
10. Training of Personnel (*Low Priority*).

SCAPA does not require retroactive SQA activities for existing software products (i.e., legacy software) that were developed under SQA requirements that were in place at the time these software products were designed, developed, tested, and deployed. In some cases extensive SQA documentation is no longer available for review because of the limited documentation retention capabilities and expectations that existed in the past. Instead, SCAPA provides guidance for ongoing SQA activities during the remaining operation, maintenance, and retirement phases of the legacy software lifecycle. These activities focus on five SQA work activities: configuration management, software design and implementation information, verification and validation testing, problem reporting and corrective action, and training of personnel.

TABLE OF CONTENTS

1. INTRODUCTION	4
2. BACKGROUND	8
2.1 Development of DOE Order 414-1C and DOE Guide 414.1-4	8
2.2 Overview of SQA Guidance for Safety Software	9
2.3 Alleviating the Gap in SQA Guidance	10
3. SQA BASICS FOR SAFETY-RELATED SOFTWARE	12
3.1 DOE Policy and SQA	12
3.2 Balancing Technical and SQA Issues	12
3.3 SQA Project Life Cycle	14
3.4 SQA Documentation Needs	14
3.5 Applicability of SQA Guidance to Legacy Software	15
4. DESCRIPTION OF SQA WORK ACTIVITIES	18
4.1 Software Project Management and Quality Planning	18
4.2 Software Risk Management	19
4.3 Software Configuration Management	22
4.4 Software Procurement and Supplier Management	24
4.5 Software Requirements Identification and Management	26
4.6 Software Design and Implementation	27
4.7 Software Safety	28
4.8 Verification and Validation	30
4.9 Problem Reporting and Corrective Action	32
4.10 Training of Personnel	33
5. GRADED APPROACH	35
GLOSSARY	38
6. REFERENCES	42

FIGURES

Figure 2.1. The DOE Framework Space for Consequence Assessment Software Prior to the Release of the SCAPA SQA Guidance	11
Figure 2.2. The Proposed Change to the Consequence Assessment Modeling SQA Landscape	11
Figure 5.1. Grading the SQA Work Activities	36

1. INTRODUCTION

In 2005, the U.S. Department of Energy (DOE) issued DOE Order 414-1C with an objective to achieve quality assurance for all work based upon the following principles:

- (1) *That quality is assured and maintained through a single, integrated, effective quality assurance program (i.e., management system).*
- (2) *That management support for planning, organization, resources, direction, and control is essential to quality assurance.*
- (3) *That performance and quality improvement require thorough, rigorous assessment and corrective action.*
- (4) *That workers are responsible for achieving and maintaining quality.*
- (5) *That environmental, safety, and health risks and impacts associated with work processes can be minimized*

A key focus of DOE O 414-1C is to provide software quality assurance (SQA) requirements for *safety software*. DOE O 414-1C Section 7(o) of DOE O 414-1C defines safety software as falling into one of three categories:

- *Safety system software.* Software for a nuclear facility¹ that performs a safety function as part of structures, systems, or components (SSCs) and is cited in either (1) a DOE-approved documented safety analysis or (2) an approved hazard analysis per DOE P 450.4 *Safety Management System Policy*, dated 10-15-96, and the Department of Energy Acquisition Regulation (DEAR) clause.²
- *Safety and hazard analysis software and design software.* Software that is used to classify, design, or analyze nuclear facilities. This software is not part of an SSC but helps to ensure the proper accident or hazards analysis of nuclear facilities or an SSC that performs a safety function.
- *Safety management and administrative controls software.* Software that performs a hazard control function in support of nuclear facility or radiological safety management programs or technical safety requirements, or other software that performs a control function necessary to provide adequate protection from nuclear facility or radiological hazards. This software supports eliminating, limiting, or mitigating nuclear hazards to workers, the public, or the environment, as addressed in 10 CFR 830, 10 CFR 835, and the DEAR Integrated Safety Management System (ISMS) clause.

Also in 2005, DOE issued DOE Guide 414.1-4 to provide information and detailed guidance on acceptable methods for implementing the safety software SQA requirements specified in DOE O 414.1C.

¹ Per 10 CFR 830, quality assurance requirements apply to all DOE nuclear facilities. That includes all DOE sites that have radiological facilities (as defined in 10 CFR 830, DOE Std 1120, and the DEAR clause).

² Department of Energy Acquisition Regulation (DEAR) clause promulgated in 48 Code of Federal Regulations (CFR) 970.5223-1, 48 CFR 970.5204-2, and 48 CFR 970.1100-1.

Consequence assessment software is used to support a variety of emergency preparedness applications in the DOE complex.³ Some consequence assessment software products fit within the definition of safety software, and some do not. To provide SQA guidance for consequence assessment models and related software that do not fall under the definition of safety software, the Subcommittee on Consequence Assessment and Protective Actions (SCAPA) has prepared this SQA guidance document. The SCAPA guidance incorporates the key elements found in the DOE G 4141-1-4 for *safety software* but does so using an appropriately graded approach that is readily implementable by DOE's emergency management community and its software suppliers. This approach strikes an acceptable balance between the need for modeling complex environmental processes (e.g., atmospheric dispersion and deposition), timely innovation, and SQA for software products that are **not** safety software.

Before discussing the specifics of SCAPA's SQA guidance, we first need to decide how to refer to consequence assessment software that is not in the safety software category. Some in the DOE community prefer the term *non-safety software*; however, this term is ambiguous. Does *non-safety software* refer to all software that does not meet the DOE definition of safety software, or does it instead apply only to software that does not address any issue related to human safety or health? Many software products that do not fall within the DOE O 414.1C definition of safety software have applications that are in some way related to safety.

To avoid confusion, this document uses the term *safety-related software* to refer to consequence assessment software that (1) does not meet the definition of safety software and (2) may be used to address issues that have safety implications. The term *other non-safety software* is applied to software that does not meet the definition of either safety software or safety-related software.

For consequence assessment applications, safety software is used in the development of emergency action levels and protective action recommendations or to support other safety-critical decision making. Safety software products are typically fast-response atmospheric dispersion and dose assessment models that use simple, conservative algorithms and the limited data on hand to support the needs of first responders. Safety-related consequence assessment software provide supplementary information that is used to better understand an emergency event and potential consequences but are not used to support safety-critical decision making. Examples of safety-related consequence assessment software are models used to estimate contaminant dispersion and ground deposition to determine where to best deploy field monitoring teams or to estimate contaminant exposures at locations that a conservative safety software model has already indicated exposure levels are below levels of immediate concern for human safety. Safety-related consequence assessment software often employ more complex algorithms, require more comprehensive input data and user training, and take longer to set up and run than safety software products.

³ The term "DOE complex" refers to all DOE and National Nuclear Security Administration (NNSA) sites and facilities.

Without its own standard set of SQA guidance, consequence assessment software developed to support safety-related and other non-safety applications have been developed and implemented using SQA guidance that differs from site to site and may even differ from project to project within a given organization. The lack of consistent guidance has created uncertainty in the DOE software development and user community. It also has generated uncertainty for those who are tasked with software review and oversight responsibilities. Questions that have been asked include

- What SQA requirements and guidance exist for the developers of safety-related software and other non-safety software?
- How should the SQA requirements and guidance developed for safety software inform and guide the development of SQA guidance for other categories of software?
- How can software developers strike an appropriate balance between model complexity, timely innovation, and SQA?
- How can users be assured that an appropriate level of SQA was applied to, and is maintained for, software acquired from others for DOE safety-related applications?
- Which SQA activities are the responsibility of software developers, and which are the responsibility of software users?
- When and how should SQA be applied in each stage of the software life cycle?

The SCAPA SQA guidance presented in this document is designed to address these questions and provide a firm foundation for the development, acquisition, and use of consequence assessment software that does not fall into the safety software category. By extension, this document can be used also to provide SQA guidance for safety-related and other non-safety software products that are used for emergency management applications but are not classified as consequence assessment software. These emergency management software products may include meteorological, hydrological, geospatial, data access, data processing, and graphical display applications.

The application of SCAPA SQA guidance does not supplant any applicable Quality Assurance Program (as required by DOE O 414-1C for each DOE organization). All applicable requirements of an organization's Quality Assurance Program must be implemented before or coincident with the application of SCAPA's SQA guidance. The SCAPA SQA guidance may be used in conjunction with other applicable SQA guidance (e.g., current or future guidance for non-safety software developed by the DOE Energy Facility Contractors Group or the American Society of Mechanical Engineers [ASME]), provided that it is not used to justify a reduction in SQA from what is specified here or in other applicable guidance documents or standards.

In this document, Section 2 provides background information on SQA. This includes the historical context for the development of the DOE SQA Order and Guide for safety software, an overview of the SQA program for safety software, and an illustration of the gap in SQA guidance for safety-related and non-safety software that existed after the release of the DOE SQA Order and Guide. Section 3 covers the need to balance

technical innovation (i.e., technical quality) with SQA, the software life cycle, the need for SQA documentation, and applicability of the SCAPA SQA guidance to legacy software. Section 4 presents the SCAPA SQA guidelines and priorities for each of the ten categories of SQA work activities. Section 5 covers the relative weighting of the ten work activities to support a graded application of these SQA elements.

2. BACKGROUND

In this section, background information is presented on the development of SQA guidance for safety software, an overview of SQA work activities for safety software, and the SQA gap that would be filled by implementing SCAPA SQA guidance for safety-related and other non-safety software.

2.1 Development of DOE Order 414-1C and DOE Guide 414.1-4

Computers routinely play a role in safety applications at nuclear facilities across the DOE complex. Over the past decade or more, SQA reviews and audits have focused DOE's attention on potential SQA problems associated with the use of safety software. This includes the accuracy of the process used to make safety decisions, the quality of the software used to design or develop safety controls, and the proficiency of personnel using safety software.

The DOE has recognized the need to establish rigorous and effective requirements for the application of quality assurance programs to safety software. After evaluating Defense Nuclear Facilities Safety Board (DNFSB) Technical Report-25 (DNFSB, 2000) and DNFSB recommendation 2002-1 (DNFSB, 2002), and assessing the state of safety software, DOE concluded that an integrated and effective SQA infrastructure was needed for safety software at all DOE nuclear facilities. To accomplish this, DOE developed and released

- Implementation Plan for Defense Nuclear Facilities Safety Board Recommendation 2002-1, *Quality Assurance for Safety Software at Department of Energy Defense Nuclear Facilities* (DNFSB, 2002)
- DOE Order 414.1C, *Quality Assurance* (2005). This is the SQA order for safety software.
- DOE Guide 414.1-4, *Safety Software Guide for Use with 10 CFR 830 Subpart A, Quality Assurance Requirements, and DOE Order 414.1C, Quality Assurance* (2005). This document was developed to provide guidance on establishing and implementing DOE Order 414.1C. The guide incorporates software application practices documented in national and international consensus standards and processes currently in use at DOE facilities. The guide is intended to be used by organizations to help determine and support the steps necessary to address possible design or functional implementation deficiencies in safety software and thereby reduce operational hazards-related risks to an acceptable level.

2.2 Overview of SQA Guidance for Safety Software

As discussed in DOE Guide 414.1-4, the goal of SQA for safety system software is to apply the appropriate quality practices to ensure the software performs its intended function and to mitigate the risk of failure of safety systems to acceptable and manageable levels. SQA practices are defined in national and international consensus standards (e.g., ASME NQA-1-2000, *Quality Assurance Requirements for Nuclear Facility Applications*.)

Safety software can come in a variety of forms. These include

- custom-developed. This is software developed for a DOE application or another governmental organization. The software may be developed by DOE or one of its contractors.
- configurable. This is commercially available software or firmware that allows the user to modify the structure and functioning of the software in a limited way to suit user needs. This sort of software may be used in meteorological data loggers or programmable logic controllers.
- acquired. This category includes commercial off-the-shelf (COTS) software, freeware, and firmware.
- utility calculation. This is typically COTS spreadsheet applications, such as macros used to perform functions in Microsoft Excel.
- commercial design and analysis software. This software is used in conjunction with design and analysis services provided to DOE from a commercial contractor.

These types of software are defined in more detail in DOE Guide 414.1-4 Section 2.1. Different safety software grading levels are provided for the five classes of software based on the software's application. Three different grading levels are provided (i.e., Level A, B, and C), and these are described in DOE Guide 414.1-4 Section 2.2. DOE Guide 414.1-4 uses the grading levels and the software types to recommend how SQA work activities are applied.

Using the various grading levels and the safety software types, DOE Guide 414.1-4 provides guidance on the software quality work activities to ensure that safety software performs its intended functions. The work activities are

1. Software Project Management and Quality Planning
2. Software Risk Management
3. Software Configuration Management
4. Procurement and Supplier Management
5. Software Requirements Identification and Management
6. Software Design and Implementation
7. Software Safety

8. Verification and Validation
9. Problem Reporting and Corrective Action
10. Training of Personnel.

All software, regardless of whether it is safety, safety-related, or other non-safety software, should be controlled in a traceable, planned, and orderly manner. The ten SQA work activities proposed in DOE Order 414.1C for safety software are also applicable for safety-related and non-safety software—except their application should follow a graded approach that is appropriate for achieving the proper balance between technical quality, timeliness of product development, and SQA.

2.3 Alleviating the Gap in SQA Guidance

After the publication of DOE Order 414.1C and DOE Guide 414.1-4, the DOE’s expectations for the SQA of safety software were established; however, no such firm guidance existed for safety-related and other non-safety consequence assessment software. Figure 2.1 illustrates this situation. It shows that specific DOE SQA requirements/guidance is provided for safety software. Safety-related and other non-safety consequence assessment software are covered by guidance that may be imposed by individual sites or DOE contractors. However, this guidance may vary significantly from site to site, from contractor to contractor at a site, and from program to program under a given contractor. This guidance may incorporate changes in widely accepted national and international standards, or it may be based on long-standing corporate “good business practices.” It may complement DOE Order 414.1C, or it may employ a markedly different framework. It may involve a formalized review structure involving internal or external audits, or it may not involve any formalized review.

Figure 2.2 illustrates what SCAPA is proposing—the introduction of SQA guidance for safety-related and other non-safety consequence assessment applications. This would constrain the DOE Order and Guide for safety software into a more defined area (i.e., avoiding creep into the safety-related and other non-safety arenas). It would establish a complementary, graded SQA framework for safety-related and other non-safety consequence assessment software that could be consistently applied throughout the DOE consequence assessment modeling community (including both software developers and end users). The SQA guidance proposed here could be supplemented by any site- or project-specific SQA requirements and by more detailed site-, company-, or project-based guidance that expands on the framework provided by SCAPA.

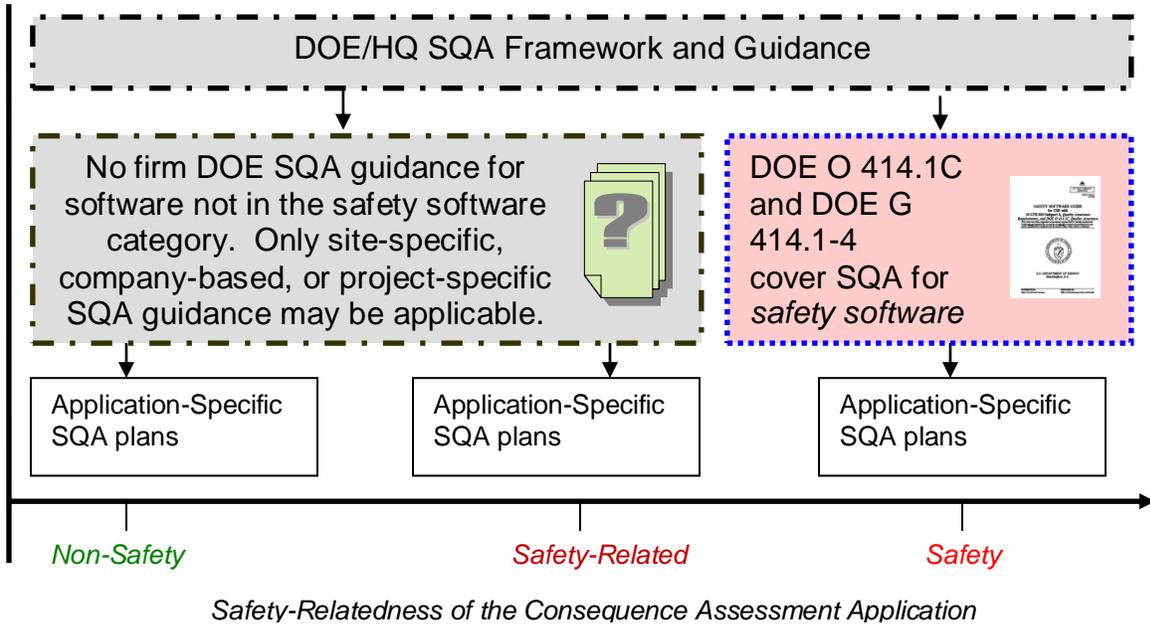


Figure 2.1. The DOE Framework Space for Consequence Assessment Software Prior to the Release of the SCAPA SQA Guidance

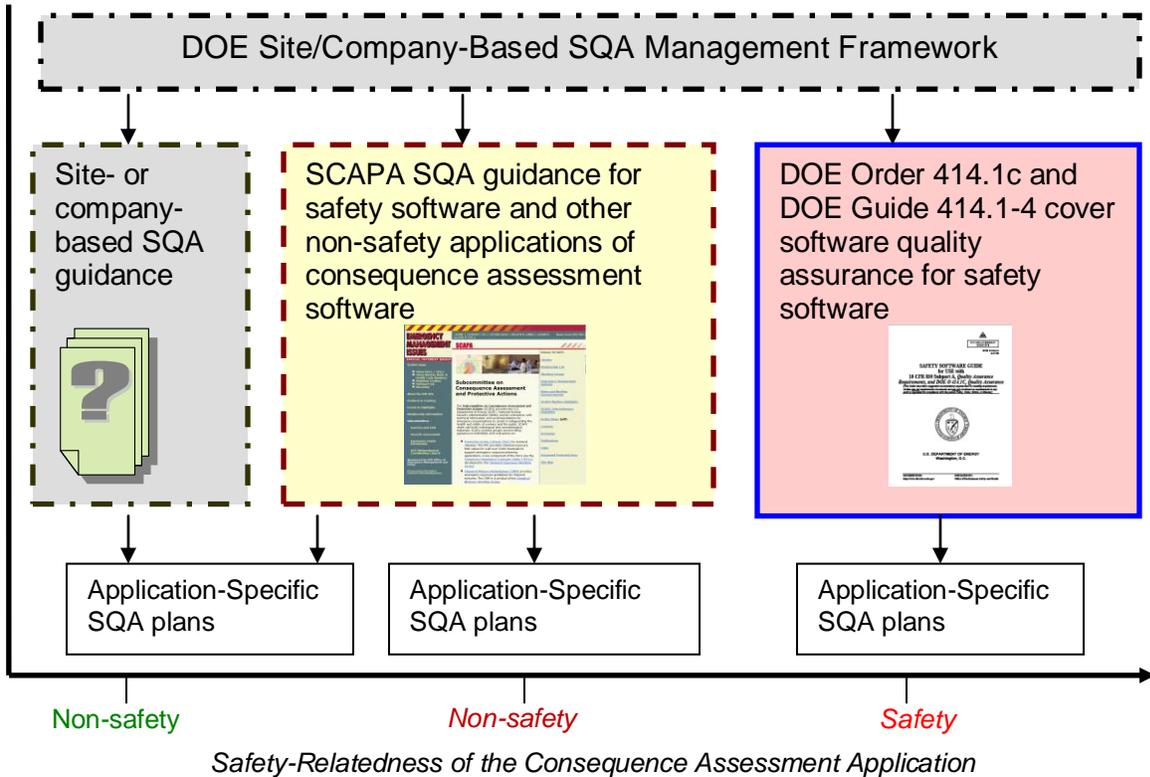


Figure 2.2. The Proposed Change to the Consequence Assessment Modeling SQA Landscape

3. SQA BASICS FOR SAFETY-RELATED SOFTWARE

3.1 DOE Policy and SQA

[DOE P 450.1, Safety Management System Policy](#), states

The hallmark and highest priority of all our activities is daily excellence in the protection of the worker, the public, and the environment. Fundamental to the attainment of this vision are personal commitment, mutual trust, open communications, continuous improvement and full involvement of all interested parties.

The following are some of guiding principles discussed in DOE P 450.1 that support the development of SCAPA's SQA guidance:

- *“Manage and conduct a consistent approach to environment, safety, and health across the DOE complex.”*
- *“Utilize innovative and effective approaches to risk identification and management.”*
- *“Apply a systematic approach to all activities that affect environment, safety, and health.”*
- *“Continue to improve our environment, safety, and health performance.”*

By calling for a consistent and systematic approach across the DOE complex for dealing with environment, safety, and health, DOE P450.1 provides a strong driver for providing consistent SQA guidance for all of the complex's consequence assessment software. By calling for an innovative and effective approach for risk identification and continued improvement in performance, DOE P450.1 also provides a strong driver for developing models that exhibit the technical complexity needed to model complex environmental processes.

3.2 Balancing Technical and SQA Issues

When consequence assessment or other emergency preparedness software is being developed, there is a need to balance a number of different and sometimes competing considerations. These considerations include

- Environmental processes are complex and can be difficult to model. Modeling complex processes such as atmospheric transport, diffusion, and deposition involves the development and incorporation of sophisticated algorithms and the need for copious amounts of environmental data (e.g., wind measurements at multiple monitoring locations and heights above the ground, geospatial data, particle-size distributions for contaminant materials). These more sophisticated models can abandon many overly conservative, simplifying assumptions and do a

more accurate job simulating real-world environmental conditions. On the downside, the more complex the model, the greater the size of the code and the more data required to support the simulations.

- Timely innovation involves incorporated improvements into the modeling system and making these enhancements available to the users without undue delay. Unfortunately, documentation and testing activities require time and resources that can delay the release of new software.
- SQA focuses on ensuring that software appropriately implements and carries out its design.

While not mutually exclusive, the development of complex environmental modeling software, the timely innovation and release of new software, and the incorporation of rigorous SQA often compete for the same limited set of project resources. An overemphasis on one element at the expense of other priorities can be a recipe for failure. For example, insufficient SQA can result in the release of poorly verified, validated, and documented software that may lead to the misuse of the product or generation of inaccurate information. Alternatively, too much SQA can stifle technical innovation and delay the release of much-needed software enhancements, forcing DOE personnel to use underperforming or inadequate software that can compromise the accuracy of their assessments. An appropriate SQA program must consider both the benefits and costs of SQA and achieve an appropriate balance that ensures adequate quality without unduly reducing modeling capabilities or functionality or unduly delaying the release of needed products.

The ability of safety software to provide consistently conservative and repeatable estimates of consequences that never (or almost never) underestimate potential impacts is paramount over the ability of the software to accurately estimate consequences across the modeling domain. In contrast, safety-related and non-safety consequence assessment software typically do not focus on worst-case impacts. For these software products, fidelity in modeling complex real-world processes and the timely release of model enhancements may be a higher priority for achieving assessment objectives. Some uncertainty in regards to the design and operation of the safety-related and other non-safety software can be accepted if the software can be shown (through validation and verification activities) to be producing, on the whole, more realistic estimates of environmental and health consequence. As a result, when compared to safety software, safety-related and other non-safety software products tend to have an optimal resource allocation and requirements balance point that is incrementally shaded more toward technical complexity and timeliness than SQA.

The application of the DOE SQA guidelines for safety software to other types of software could impose excessive SQA requirements that would tend to stifle technical innovation and delay the release of new software. In the long run, this situation could curtail the effectiveness of DOE consequence assessment tools. The SCAPA SQA guidelines are intended to avoid a situation where the development and maintenance of safety-related and other non-safety consequence assessment software is held back by excessive SQA requirements. It is also intended to avoid the opposite situation, where the lack of

applicable SQA guidelines allows ad hoc and potentially inadequate SQA programs to govern model development and maintenance.

In DOE Emergency Operations Centers (EOCs), SCAPA advises that emergency managers and consequence assessors employ both simple consequence assessment models that meet the SQA requirements of DOE Order 414-1C for safety software and more sophisticated models that would meet the SCAPA SQA guidance for safety-related software. The safety software models should be used to make timely initial assessments of maximum exposure to protect worker safety and health. More complex safety-related models should be used to estimate atmospheric dispersion and deposition for more complex safety-related analyses—for example, to estimate exposures and deposition values at distances downwind where a more sophisticated and accurate treatment of atmospheric spatial and temporal variability is needed, especially at sites with complex terrain. Results from these more sophisticated models cannot be used to supersede safety recommendations that were based on projections from simpler safety software models.

3.3 SQA Project Life Cycle

All software has a life cycle. This life cycle can be divided into a series of phases, beginning with the development of the software concept and ending with the retirement of the software product. These discrete life-cycle phases are

- Concept
- Requirements
- Design
- Implementation
- Test
- Installation, Checkout, and Acceptance Testing
- Operations
- Maintenance
- Retirement.

SQA work activities such as configuration management and training are conducted across all phases in the software life cycle. Some SQA work activities, such as validation testing and procurement management, may only not be pertinent in some life-cycle phases. The SCAPA SQA guidance is designed to be applicable through the entire software life cycle—from the initial concepts and requirements development phases; through software design, implementation, and testing phases; and into the operations and maintenance phases.

3.4 SQA Documentation Needs

The amount of material needed to provide adequate documentation for each work activity is dependent on the requirements and scope of the individual project. Simple projects, such as those involving minor upgrades to simple consequence assessment models, may

require minimal documentation to address the SQA work activities. Large projects, such as those involving extensive modifications to complex models (e.g., with many thousands of lines of code), may require a substantial amount of documentation to maintain an appropriate level of SQA. The scope of the documentation needed for each specific category of SQA work activity is described in Section 4.

Documentation is typically needed for each of the ten work activities. Some of these documentation products may be as short as a page or two, while others might be very extensive. These documentation products do not necessarily need to be stand-alone documents. At the discretion of project management, SQA documentation may be combined into a smaller set of documents with each covering one or more categories of the SQA work activities. The number of SQA documents is not important; what is important is that each SQA work activity be documented in sufficient detail and that project participants know where to find this documentation in either a stand-alone document or as part of a documentation package that covers several SQA work activities.

3.5 Applicability of SQA Guidance to Legacy Software

Throughout the DOE complex, a number of relatively sophisticated consequence assessment models have been successfully and reliably used and tested over a period of many years. These software products were developed under the SQA requirements that were spelled out in their original scopes of work and project management plans. For most, this involved following a comprehensive set of good business SQA practices for model development, documentation, training, and testing.

In many cases, extensive SQA documentation (e.g., design documents, user's guides, verification and validation [V&V] testing) exists. In other cases, limited records retention periods, degradation of old electronic storage media, and limited storage space for hardcopy records have resulted in the permanent loss of some SQA documentation (e.g., original model V&V test data sets and results). In today's technological environment, when electronic memory is ubiquitous and inexpensive and electronic files never have to be purged, it is hard to recall that current data storage capabilities weren't readily available in the recent past. Many project records and testing files were only kept for a limited period (e.g., five years) before records disposition schedules called for their disposal. This was especially true for hardcopies of testing records that were generated in the 1980's and 1990's. Without adequate electronic storage capabilities, space was frequently unavailable to store all of the hardcopy products associated with extensive V&V testing programs.

Given DOE's many important missions, and limited financial resources available for consequence assessment model development and maintenance, it makes little sense from a cost-benefit standpoint to go through the expensive and time-consuming process of recreating discarded SQA documentation for time-tested "legacy" codes. It makes even less sense to stop using time-tested models that are not used for nuclear safety applications simply because the original SQA information from the pre-operational phases of the software lifecycle were not retained.

Discarding such models would result in either a significant drop in the consequence assessment modeling capability of DOE programs (e.g., replacing sophisticated modeling systems with less accurate, more simplistic systems just because they have all of their SQA documentation) or require an expensive transition to, or the development of, a brand new modeling system that would at best offer only comparable capabilities to the existing modeling system.

As a result, SCAPA has elected to implement a grandfather clause⁴ to allow the continued use of existing consequences assessment software products that have adequate technical and user documentation, an extensive history of use in the DOE complex (or for other government agencies), and a program to maintain appropriate SQA throughout the software's operations, maintenance, and retirement life cycle phases. For a legacy code to remain in compliance with SCAPA's SQA guidance all future development work (e.g., software updates and enhancements) must fully comply with the SQA guidance for all ten SQA work activities. In other words, all modifications to existing portion of the legacy code must be designed, implemented, documented, and tested in manner that is equivalent to what would be expected with the development of new software.

3.6 Extent of SQA Guidance – Defining the Limits of Applicability for a Software Product

Consequence assessment software often make use meteorological, environmental, geospatial, and other data that are generated or acquired from other sources. The question arises as to whether quality assurance guidance outlined in this document must be applied by consequence model developers to the software that generates these supporting data. The following guidance is intended to guide software developers and custodians in answering this question:

- All software that is to be contained within the consequence assessment model is subject to SCAPA's SQA guidance.
- All procured software that is used with the consequence assessment model is subject to the SCAPA SQA guidance for procured software. This would include software used to read input data, process input data, and produce graphical or numerical output products.
- SQA requirements for software that is not part of the compiled consequence assessment software, and is used to provide data for the consequence assessment software, is the responsibility of the software user and not the software development and custodial team. For example, the consequence assessment model development team is responsible for the SQA involving accessing, processing, and using meteorological data within their software. They are not responsible for the SQA of the software involved in measuring meteorological

⁴ A grandfather clause is an exception that allows the continued use of existing products when new standards or requirements will apply to all new products or the future modifications of existing products.

parameters, pre-processing the collected data, and processing the data into its final format if they did not create that software. The software user (e.g., the project that is using the meteorological data) is responsible for ensuring that appropriate SQA is in place for the meteorological software that provides data to their consequence assessment model. This SQA requirements for the software that generates supporting data should be at least as stringent as is required for the consequence assessment model. In some instances this supporting software may need to meet safety software standards as it supports not only safety-related consequence assessment models but other safety software applications.

4. DESCRIPTION OF SQA WORK ACTIVITIES

In this section, the ten SQA work activities are described, and minimum levels of compliance are specified to meet expectations for safety-related consequence applications. A priority rating is assigned to each work activity to illustrate the relative importance of the activity in the SQA program. The prioritization of work elements is discussed in Section 5. For many software products, additional SQA beyond the minimum levels described below will be warranted. Additional SQA may be required to meet programmatic requirements, organizational requirements, and client or customer expectations. Guidance on additional SQA measures are provided in standards and guidance documents such as DOE Guide 414.1-4 and ASME NQA-1-2000

4.1 Software Project Management and Quality Planning

Summary: A high-level software project management plan or a software development plan is developed to describe the framework for software development, modification, and testing activities. The plan should identify involved software components, spell out tasks, and describe SQA requirements. This framework is prepared at the beginning of the project and is maintained throughout the project life cycle.

Priority Rating: High.

Software project management and quality planning are key elements in establishing a foundation to ensure that a software product meets all the goals and objectives for development, modification, and testing activities. At the start of any project or task that will develop a new, or modify an existing, software product, a high-level software project management and quality planning document is required. This document may take the form of a software project management plan (SPMP), a software development plan (SDP), or similar document.

Typically an SPMP or SDP are the controlling documents that define and guide the activities necessary to satisfy project software requirements, including the SQA requirements. At a minimum, software project management and quality planning needs to

- Identify the software products or components that are the subject of this project.
- Identify and characterize the software development, modification, and maintenance tasks. These characterizations should
 - Cover the objective of each task.
 - Describe the key activities of each task.
 - Present task management assignments.
 - List key milestones and scheduling information (e.g., start and end dates) for each task.
 - Outline resource and staffing requirements.

- Discuss key dependencies and assumptions for each task.
- Identify and describe all applicable SQA documentation requirements.

These plans should be initiated early in the project life cycle and be maintained throughout the life cycle of the software project. Updates should be made to the planning document to reflect major changes in the project.

For legacy software, the Software Management Plan should cover these minimum requirements by focusing on the remaining phases of the software life cycle (e.g., operations, maintenance and retirements). The tasks to be identified and described are those associated with maintaining configuration management, software testing, user and technical documentation, problem reporting, and training. In some cases, software project management and quality planning information may be embedded in the overall project planning document. Regardless of whether embedded in a larger planning document or put into a stand-alone document, software project management and software quality planning activities must be documented.

4.2 Software Risk Management

Summary: Before any work to develop or modify software is begun, the risks associated with this effort and the specify actions that will be taken to address or mitigate these risks are documented. The documentation is updated if software risks or management approaches change significantly over the course of the project.

Priority Rating: Medium.

In general, risk is the product of the (1) likelihood (or probability) an adverse incident will occur and (2) the consequences resulting from that incident. Risk management is the process of assessing risks and taking steps to reduce risks to an acceptable level. Risks can be addressed by reducing the likelihood of occurrence of adverse events or by reducing the consequences that could result from adverse events.

Software risk management provides a disciplined environment for assessing characterizing risks, tracking risks, and resolving risks. Risk assessment involves the identification of adverse incidents that could jeopardize the successful completion of the software project, the characterization of the likelihood of adverse incidents, the identification and characterization of potential consequences to the project from adverse incidents, the estimation of risks, and the identification of key contributors to overall risk.

Risk resolution can involve risk avoidance, mitigation, and risk transference activities. Risk avoidance involves taking steps to reduce the likelihood of an adverse event. Risk mitigation involves taking steps to reduce the consequences of an adverse event. Risk transference involves reducing negative consequences through the use of insurance or other risk-sharing method.

Risk tracking involves the documentation of risk-related information, risk resolution processes, the reduction in risk that occurs from the implementation of risk management activities, and level of residual risk that exists after risk management efforts are implemented. This level of residual risk needs to be accepted by the project.

Software risks should be assessed and risk resolution measures planned before any software development work begins. The assessment, resolution, and tracking of the risks should continue throughout all phases of the project's life cycle. When evaluating risks, the project team should be mindful that a relatively small adverse impact during one phase of the software life cycle has the potential to trigger a much larger adverse impact during a subsequent phase in the application's life cycle. In addition, risk resolution activities taken to address one issue could create a new set of risks that may need to be evaluated.

Risk management can be accomplished using a qualitative, semiquantitative, or quantitative approach. A qualitative approach may assess risks using two or three subjective risk categories (e.g., high, medium, or low). Qualitative assessments are easy to conduct and provide a concise analysis; however, results from such an assessment may be too simplistic to effectively manage software risks for some projects. Quantitative approaches involve evaluating risks in terms of estimates of the likelihood of an adverse incident (e.g., likelihood of occurrence of the adverse event in a given year) and impacts (e.g., dollars lost, weeks a project is delayed). The difficulty in performing quantitative analyses and the high uncertainties in their results effectively restrict the use of quantitative analyses. A semiquantitative approach lies between a qualitative and quantitative approach. This approach may use more evaluation categories than a qualitative approach to achieve greater resolution in risk characterization (e.g., ranking risks on scale of 1 to 10) without requiring the specificity of a quantitative approach. Each of the three approaches has its advantages and disadvantages; therefore, the appropriate approach for evaluating risks and making risk management decisions is left to the discretion of the project manager.

Examples of potential software risks that have been identified for safety software and might be applicable to some safety-related software include

- incomplete or volatile software requirements
- specification of incorrect or overly simplified algorithms
- hardware constraints that limit the design
- unexpected interactions with or reliance on unknown systems
- potential performance issues with the design
- a design based upon unrealistic or optimistic assumptions
- design changes during coding
- incomplete and undefined interfaces
- using unproven computer and software technologies such as programming languages not intended for the target application
- inexperienced software developers
- new versions of the operating system

- unproven testing tools and test methods
- insufficient time for development, coding, and/or testing
- undefined or inadequate test acceptance criteria
- potential quality concerns with subcontractors or suppliers.

Risk management is applicable for both new software development efforts and model enhancement activities. It is acceptable to undertake a high-risk software development project if the benefits of a successful outcome outweigh the risks of failure. Missteps, false starts, and unsuccessful projects are part of the scientific experience, and the risk of failure is often acceptable. Lessons learned from the partial or complete failure of consequence assessment modeling projects have spurred the development and contributed significantly to modeling breakthroughs. From an SQA perspective, it is important to characterize the risk of failure, take steps to minimize these risks, and appropriately recognize failures when they occur. However it is unacceptable from an SQA standpoint to go into a project without understanding the risks, fail to recognize adverse situations during risk planning, and release a seriously flawed modeling product.

When evaluating the risks of proceeding with a software development or modification project, it is appropriate to consider the risks associated with not proceeding with the project. In some cases, the primary alternative to the project may be to continue using an existing software product that does a poor job modeling real-world conditions because of overly-simplified or inaccurate algorithms or the inadequate use of available environmental data. The risks associated with continuing to use the existing product may outweigh the risks associated with developing and implementing a new consequence assessment model.

At a minimum, the software risk management work activity should document

- potential adverse incidents that can impact the project budget, resources, schedules, and technical goals of the project
- the relative likelihood of these adverse impacts occurring
- the potential consequences of these adverse impacts on the project and product
- methods to reduce these risks through risk avoidance, mitigation, or transference
- the accepted level of residual risk.

Further guidance on SQA risk management can be found in ASME NQA-1-2000 and IEEE Standard 16085-2004.⁵ SQAS21.01.00-1999, *Software Risk Management: A Practical Guide*, discusses a risk taxonomy, risk transference, and risk avoidance.

For legacy software that is not undergoing any modification, the development of a software risk management assessment is generally not needed. However, if there is a substantial change in site practices, threat profiles, or other parameters that can appreciably change the risk profile associated with the use of the legacy software product,

⁵ International Organization for Standardization (ISO)/Institute of Electrical and Electronics Engineers (IEEE) Std 16085, *IEEE Standard for Software Engineering: Software Life Cycle Processes—Risk Management*, IEEE, 2004.

the need for a software risk management study and plan should be reassessed and appropriate risk management documentation developed.

4.3 Software Configuration Management

Summary: A software configuration management plan is implemented to provide software version identification; the documentation, control and approval of software changes; status tracking; secure maintenance/storage of baseline software; and assignment of configuration management responsibilities to designated staff.

Priority Rating: High.

The software configuration management activity identifies the functions and tasks required to manage the configuration of the software product. Configuration management is of great importance. It is used to identify the current operational version of all software components and distinguish them from older or experimental software components. It is used to maintain the current operational version of the software in secure and backed-up locations so that the software is safeguarded in the event of a natural or man-made disaster. It is used to ensure that all changes to a software component and all releases of the software are conducted by authorized personnel, that changes to the software are appropriately labeled within the software, and that all system documentation is appropriately modified to reflect all software modifications.

The following three areas should be addressed when preparing and implementing a configuration management plan for the design, implementing, testing, and maintenance of a software product:

- configuration identification. A software configuration baseline should be defined at the completion of the software design, implementation, testing, and operation phases of the software life cycle. A labeling system is to be used to uniquely identify each software component (e.g., program, subroutine) and distinguish it from earlier versions of the component. Each new version of the overall software product is to have a unique version number (e.g., Version 2.3.1.12) to distinguish it from previous versions of the product in the software baseline. The adopted configuration identification system is to be maintained throughout the life cycle of the development product. If a new identification system is developed during a project life cycle, its relationship to any previously used identification systems must be documented.
- configuration change control. Changes to the software made during each stage of the software life cycle are to be documented, evaluated, and approved prior to inclusion in the software baseline. The documentation should include a description of each change and an identification of where the change occurs in software. In addition, it is recommended that the rationale for the change also be provided. Before inclusion in the software baseline, all changes to the software

should be tested (following all appropriate testing requirements) and then reviewed and approved by the person or team responsible for controlling the baseline version of the software product. Modifications should be made to all software documentation (e.g., design document, user's guides) to capture changes to the software.

- configuration status control accounting. The status of all modified software components is to be maintained and tracked until the modified software is accepted for incorporation into the baseline version of the software. Status information is to be recorded to identify the status of proposed changes to each software component and to support configuration identification and change control activities.

The configuration management plan should include documentation of the configuration management roles of the software team. This includes the roles and responsibilities of model developers, testing personnel, documentation writers, the code custodian, and the project manager. The names of the team members playing key project roles should also be documented in the configuration management plan or an associated team assignment list. All members of the team are accountable for implementing the required configuration management procedures and practices in their respective areas of responsibility.

For all phases of the software life cycle, the configuration management plan must describe a process for safely backing-up and storing the software baseline and maintaining appropriate expertise on the project team. The software baseline should be backed-up and stored in multiple locations with an appropriate geographic separation so that a natural or man-made disaster does not permanently disrupt software integrity, configuration management history, or the team's ability to access and modify the code. In addition, the configuration management plan should include a plan for team member succession and knowledge transfer, so that the unanticipated loss of key team members (e.g., change in employment, unexpected change in health status) does not result in an unrecoverable loss of software knowledge or expertise.

A commercial software version control product is often a convenient tool to support an effective configuration management process. Such tools can be used to control alterations to software components, protect against the inadvertent loss of files, track changes made by different team members, and record when specific changes were made. Such tools can limit which team members can check in and out individual software components and allow project managers to undo changes and easily roll back to earlier versions of a software component.

The configuration management system should be reviewed periodically to ensure that configuration management guidelines are appropriately followed. The software custodian or project manager should have the option to schedule configuration audits and reviews as needed. Some software products may warrant third-party audits, appraisals, or reviews to verify that all configuration management requirements are being met.

Physical configuration audits and functional configuration audits are examples of audits or reviews that might be performed.

At a minimum, the configuration management work activity should involve

- a configuration identification system. This system establishes a software configuration baseline and uses a labeling system that uniquely identifies each new version of the overall software product and identifies each software component from earlier versions of that component. The adopted configuration identification system is to be maintained throughout the life cycle of the development product.
- a configuration change control system. This system should document each change made in software. Before inclusion in the software baseline, all changes to the software should be tested and approved by the designated change approval authority. Software design and user documentation should be modified to capture all significant changes to the software.
- a configuration status control accounting system. The status of all modified software components is to be maintained and tracked until the modified software is accepted for incorporation into the baseline version of the software.
- a clearly defined set of configuration management roles and responsibilities. All key roles and responsibilities should be documented and maintained throughout the software life cycle.
- a back-up and secure storage system for the software baseline.

For legacy software, a configuration management program should at minimum provide documentation of the current software baseline and provide guidance through all future phases of the software life-cycle. Of particular note is the need to adequately back-up and securely store the software baseline. The roles and responsibilities for maintaining configuration management should also be documented.

4.4 Software Procurement and Supplier Management

Summary: Before software tools or components are procured from outside sources, a plan for addressing software procurement and supplier management issues is implemented. The plan should include an evaluation of the quality of the SQA of the procured software. If the software's SQA cannot be adequately evaluated or does not fully meet project standards, V&V testing and risk analysis/management activities can be conducted to determine whether the product can be used.

Priority Rating: Medium.

Many software projects will have procurement activities that require interactions with contractors and suppliers. Procurement activities may be as fundamental as the purchase

of tools for software development (e.g., text editors) or as complicated as procuring key software components (e.g., graphics engines) for inclusion in the software product's baseline. A variety of acceptable approaches for software procurement and supplier management are available to the software development team. The project team should establish acceptance criteria for software procurement.

It is recognized that many factors go into the selection of a software product from a vendor. In addition to SQA, this includes technical quality, speed, cost, ease of use, and supplier reputation. A review of the SQA of a software product that is a candidate for procurement should include a review of available SQA-related information. At a minimum, this information should include

- documentation of the software design and the technical underpinning of the software
- user's guides
- documentation on the system for notifying users of product defects, new releases, or other issue that may impact usage of the software
- instructions on how users report software issues to the vendor.

It should also include, as warranted,

- product testing information and V&V data sets
- instructions on how a user can request technical assistance/guidance from the supplier of the procured software.

The evaluation of a supplier's SQA program can be streamlined if the supplier is able to provide a declaration or certification/accreditation that the supplier's software meets recognized SQA standards. For legacy software, software procurement and supplier management may not need to be addressed unless there is a change in procured software (e.g., incorporation of a new or modified component into the consequence assessment software) or a significant issue has been raised regarding the quality of the existing, procured software.

Many vendor products (e.g., operating systems, text editors, compilers, spreadsheets, database tools, graphics engines) from reputable software developers have extensive SQA programs that do not require much, if any, independent verification. Similarly, widely used products from large companies, with an extensive set of documentation, a large user community, forums for users to share feedback on the software and report problems, and active programs for providing software updates database may require only a cursory review of their SQA program.

If there are unresolved questions about the SQA of a supplier's software product, the project can compensate for this information gap by conducting its own V&V testing of the product. This should be followed up by conducting a risk assessment/management analysis regarding the use of the software. If the risks of using this software are understood and can be reduced to an acceptable level, the software may be used to support the project. This applies to both COTS and legacy software (including software

developed in-house) that was developed under an SQA program that does not meet current project standards or whose SQA documentation has not been retained.

4.5 Software Requirements Identification and Management

Summary: Software requirements should be identified and documented prior to beginning software design. Software requirements should be managed throughout the software life cycle.

Priority Rating: Medium.

Before the design phase of software development (including the creation of new software products and the modification of existing software products) begins, the key requirements for the software should be developed and documented. The identified requirements may be presented in a stand-alone requirements document or included in other project SQA documentation. At a minimum, requirements should cover areas such as software function, performance, the interface with users, and installation considerations. Other requirements that should be considered include security, reliability (i.e., potential for software failure or malfunction), and design constraints where appropriate. For legacy software that is not being modified, software requirements identification and management work activities may not be needed.

When conducting the software requirements identification and management activity, the assessment of software function should include a consideration of whether the software is of adequate technical sophistication to fulfill its designed function. For example, in evaluating whether the parameterization of the spatial and temporal variability of the wind field is appropriate for a given atmospheric dispersion model the design team should consider the software's intended use. A simple, straight-line Gaussian parameterization may be appropriate for generating rapid estimates of atmospheric dispersion at distances close to the release site. However, more sophisticated modeling approaches may be required for estimating dispersion at longer time and distance scales, particularly in complex terrain environments. In the latter case, a great level of technical complexity and the input of a more detailed set of environmental data may be required to achieve the required level of accuracy in model projections.

Requirements should be provided to the software design team and included, as needed, in procurement contracts and acquired software agreements. The level of detail and quantity of the specified requirements will be a function of the scope and size of the software project.

Once software requirements have been defined and documented, they should be managed to minimize conflicting requirements and maintain accuracy for later validation activities to ensure the correctness of the software placed into operations. Software requirements should be traceable throughout the software life cycle.⁶

⁶ ASME NQA-1-2000, op. cit., Part II, Subpart 2.7, Section 401, p. 106.

The detail and format of the software requirements may vary with the software type. Custom-developed software most likely will contain a larger number of software requirements than procured software, spreadsheet macros, and other types of software.

Some model developers and software users may rate Software Requirements Identification and Management as a high-priority work activity. That is perfectly acceptable; however, in this guidance we chose to assign a medium priority to this work activity. This slightly lower level of priority is not intended to minimize the importance of software requirements; it simply reflects that these requirements may already be covered to a considerable degree in the SPMP, SDP, or software design documentation. In addition, although not formally documented, software requirements may be captured in project notes or be implicitly understood by the software development team.

4.6 Software Design and Implementation

Summary: The documentation associated with software design and implementation provides a wide array of information on the software and how it works, including its theoretical basis, structure, requirements for operation, control and data flow. This documentation is intended to support both software developers and users.

Priority Rating: High.

The documentation associated with software design and implementation is intended to support the software development team and software users. For the development team, this documentation provides a retrievable, configuration-managed record of the software and how it works. This supports the development and maintenance of the software through both the normal and unanticipated turnover of team members and after periods of project quiescence when team members may not recall all the details of software operation and data flow. For software users, this documentation provides an explanation of what the software is doing, why it is doing it, and how it is being done. It allows the users to determine if the software is appropriate for their intended applications and how best to use the software to achieve their technical objectives. In addition, detailed instructions on how to use the software should be supplied in a user's guide.

Software design and implementation documentation is developed and documented to support technical and quality assurance reviews of the code, verify software performance, assist in configuration management, and support software users. The software design and implementation documentation should identify the operating system, function, interfaces, performance requirements, installation considerations, design inputs, design constraints, and any other assumptions related to effectively running the software. Documentation should describe how the software will interface with the users and/or other software and how the software will function internally. At a minimum, components of software design and implementation documentation for both new and legacy software should include

- a description of the major components of the software and how these components support the software's overall operation

- a description of hardware and software requirements for using the code (e.g., minimum computer processing and memory requirements, permitted operating systems, communication requirements, needed support software)
- instructions on how to install, configure, and use the software
- a technical description of the software that covers its theoretical basis (e.g., governing equations for atmospheric transport and diffusion, deposition, radiological decay, dose computation)
- a description of structure/format of the input and output data and allowable ranges for these data

For new or modified software, software design and implementation should also at a minimum provide

- a presentation and description of the control flow and control logic during the execution of the model (e.g., flow charts showing how program control moves through portions of the software and back and forth to program subroutines)
- a presentation of the data flow through the software (e.g., descriptions of where data are entered and processed within the software)
- a source code listing that includes comments and understandable variable names
- definitions of key parameters used in the code.

Some design and implementation information may be intended primarily for use by the software development team, but a lot of information should be packaged in a format so that it is available to all software users to support their evaluation and use of the software.

The software design description may be combined with the documentation of the software requirements or software source code;⁷ however, a stand-alone user's guide should be prepared for all software. For sophisticated models, more detailed design and implementation activities and documentation may be required on a case-by-case basis. Guidance on additional activities is presented in Section 5.2.6 of DOE G 414.1-4.

4.7 Software Safety

Summary: Software safety analyses and documentation are intended to identify possible hazards that have the potential to reduce software reliability and to describe methods that can be used to eliminate, mitigate, or monitor potential hazards.

Priority Rating: Low.

⁷ ASME NQA-1-2000, op. cit., Part I Introduction and Section 801.2, p. 16.

The purpose of software safety analyses and documentation is to identify potential hazards (i.e., abnormal conditions and events) that have the potential to reduce software reliability and cause the software to malfunction and produce erroneous results. Some potential problems that may need to be considered are (1) complex or faulty algorithm, (2) lack of proper handling of incorrect data or error conditions, (3) buffer overflow, and (4) incorrect sequence of operations due to either logic or timing faults.

The proper design of the software is critical to ensuring an acceptably low risk of software failure. As a result, software reliability is already an element to be considered during the development of software project management and quality planning (Section 4.1) and software requirements (see Section 4.5). Software safety documentation simply addresses the topic of software reliability in more detail and describes the methods that are (or can be) employed to eliminate, mitigate, or monitor potential hazards. Once identified, these methods may be incorporated into software design and user documentation.⁸

Software safety can be enhanced during software design by focusing attention on the principles of simplicity, decoupling, and isolation to eliminate or reduce hazards.⁹ An increasing complexity of the software design, including the logic and number of data inputs, is well correlated with an increasing likelihood of defects in software components.

Potential hazards, if realized, can cause full or partial failures of a software product. Full failures are generally easy for users to identify (e.g., the program aborts and displays an error message). Partial failures can be more insidious in that they may degrade the capabilities of the software without any warning being provided to the software user. The application of some quality assurance design techniques, such as building fault detection and self-diagnostics into the software, are options that may be considered for implementation in some software products. Self-diagnostics may be implemented to detect and report software faults and failures in a timely manner and allow actions to be taken to avoid excessive impacts.

At a minimum, software safety analyses should include an identification of the potential software-related hazards that can cause the software to malfunction and the methods that are being employed to eliminate, mitigate, or monitor these hazards. For legacy software that is not being modified, a software safety analysis may not be needed.

The priority for having stand-alone safety software documentation as part of an SQA program is listed as low. That is because even if this documentation is not generated, system reliability should still be carefully considered as part of the development of system requirements, system design, implementation, and user guide documentation.

⁸ ASME NQA-1-2000, op. cit., Part II, Subpart 2.7, Section 402, p. 106.

⁹ Leveson, op. cit., pp. 400–412.

4.8 Verification and Validation

Summary: Verification and validation activities check that software is effectively implementing its technical basis and requirements and fulfills its designed purpose. These activities are conducted in many phases of the software life cycle and include reviews, observations and testing, acceptance testing, and reliability testing.

Priority Rating: High.

Verification and Validation (V&V) is one of the most important SQA activities. Verification is the process of determining if the software is meeting established requirements and doing what it designed to do (e.g., is it meeting developer requirements? Is it appropriately implementing its technical basis by executing the equations upon which its calculations are based? Is it properly reading and processing input data?). Validation is the process of determining the degree to which the software is providing an accurate representation of the real world from the perspective of the software user (e.g., even though the software may be performing as designed, is it producing results that agree with what we would expect to see in the real world? Are the results consistent with available field measurements or the results from approved models?). For consequence assessment software, validation may be accomplished by comparing the results of the software to applicable laboratory or field studies or results from other benchmark models.

Verification is performed throughout the life cycle of the safety software. Validation activities are performed after the software has been developed and may be repeated during later phases of the software life cycle. V&V activities should be performed by competent staff, including contributions or oversight by other than those who developed the item being verified or validated (i.e., independent assessments).¹⁰ There are many specific activities that can be considered and chosen as needed to support an appropriate program of V&V. At a minimum, V&V should address reviews and inspections, observations and testing, acceptance testing, and reliability testing.

Reviews and inspections are conducted to ensure that the required project activities are being properly conducted and documented. For example, to support Software Requirements Identification and Management reviews may be conducted to ensure that all of the identified software requirements are being met and that the appropriate documentation of this is in place. Observations and testing should be conducted during a number of phases in the software life cycle, including the phases for implementation; testing; installation, checkout and acceptance testing; operations; and maintenance. Observations and testing during the implementation phase should include a review of all new or modified code elements and testing of each element to ensure that it works as intended. During the test phase, the observation and testing of the code should move to a more macro focus to test performance of major portions of the code and the entire

¹⁰ ASME NQA-1-2000, op. cit., Part I, Requirement 3, Section 801.1, p. 16.

software product. Testing should at a minimum assess functional performance, but additional testing can be added to conduct time performance, stress, security, and human-factors testing.

During the installation, checkout and acceptance testing phase, acceptance testing is conducted by the user of the software. It is up to the software developers to develop and provide users with useful guidance and support materials for acceptance testing. Users' guides, user test cases, and operational profiles can be instrumental in identifying and detailing the positive test cases and procedures. When users receive a new version of a software product, predetermined and ad hoc test cases should be run to validate that the software meets user requirements and does not perform any unintended functions.¹¹

During the operational and maintenance phases of the software life cycle, reliability testing should be performed by the user on an occasional basis to detect any degradation or departure from the software's expected performance.¹² As was done for acceptance testing, test cases and guidance material should be provided to the users to assist them in running reliability tests.

At a minimum, all software testing activities should be planned and documented. All test procedures, test cases, and test results should be documented and placed under configuration management.¹³

Modern utility calculation applications such as spreadsheet programs have grown dramatically in power. The addition of macro programming languages and the ability to incorporate "add-in" programs provide users with nearly the same capabilities as codes developed with traditional programming tools. Calculations performed using applications such as commercial spreadsheet programs may be treated in either of two ways:

1. Calculation results are checked and verified in the same manner as a hand calculation. This process is generally applicable to relatively simple calculations.
2. Files containing the calculation formulas, algorithms, or macros are subject to the entire software life cycle process. This approach is generally applicable to more complex or extensive calculations and applications that are frequently used.¹⁴

Section 101.1 of ASME NQA-1-2000, Subpart 4.1, provides useful guidance on V&V of utility calculations.

For legacy software, V&V activities should focus on providing test cases and user guidance for acceptance and reliability testing.

¹¹ ASME NQA-1-2000, op. cit., Part II, Subpart 2.7, Section 404, p. 106.

¹² ASME NQA-1-2000, op. cit., Part I, Requirement 11, Section 400, p. 30.

¹³ ASME NQA-1-2000, op. cit., Part I, Requirement 11, Section 200, p. 29.

¹⁴ ASME NQA-1-2000, op. cit., Section 101.1, Subpart 4.1.

4.9 Problem Reporting and Corrective Action

Summary: Problem reporting and corrective action covers the identification and documentation of software problems, reporting, assessment, correction, and communication with users. Maintaining a robust problem reporting and corrective action process is a requirement for maintaining a reliable software product.

Priority Rating: High.

The problem reporting and corrective action system is designed to cover the identification and documentation of software problems, the reporting of problems to the software development and maintenance team, the evaluation of problems by the software team, the correction of the problems, and the effective dissemination of problem information and fixes to all software users.

Software problems may be detected by members of the software team or by a software user. Clear instructions should be provided to the user community on how to document and communicate information on problems to the software development and maintenance team. Mechanism for doing this include

- providing a structured problem reporting form with clear instructions in the software user's guide on how to submit this form to the software team
- providing problem reporting contact information within the software (e.g., a Help button that provides contact information for the code custodian)
- enabling a problem reporting function on the software website, for software products with a diverse user base. A preferred method is to have a problem reporting form that offers an automated way of submitting the problem report to the code custodian and other members of the software development and maintenance team.

All problem reporting forms should include requests for the name and contact information for the originator of the report, the date and time the problem was submitted, the version number of the code, a detailed description of the problem, and contact information for the code custodian (e.g., name, email address, phone number).

On the receiving end, roles and responsibilities should be assigned on the team for logging, evaluating, addressing, and dispositioning all problem reports. The evaluation process should cover determining whether a reported problem is valid and, if so, is this an error or an area for improvement? If an error, the team should identify where in the software the error occurs and the extent and consequences of the error. All users of the software should be notified promptly of any significant errors and the potential consequences. This notification should not be delayed until a corrected version of the software is available for release. Prompt communication with registered users should be made by an appropriate method (e.g., email, phone calls, letter). With software that has unregistered users (e.g., software transferred from a known user to a new user, software

downloaded from a website by anonymous users), information on the problem should be posted promptly on a problem reporting webpage for the software product. A similar notification process should be followed when a remedy for the problem (e.g., a corrected version of the code, instructions for a workaround) has been fashioned.

Maintaining a robust problem reporting and corrective action process is vital to maintaining a reliable and vital software product. This problem reporting and corrective action system need not be separate from the other problem reporting and corrective action processes if the existing process adequately addresses the items in this work activity.¹⁵

At a minimum, the problem reporting and correction work activity should for all software include

- development and maintenance of problem reporting and correction work activity assignments and responsibilities. This includes instructions for receiving, documenting, addressing, and promptly communicating problem reports and solutions with the development team and user community.
- development and maintenance of instructions to the software users on how to report problems identified during the installation or operational use of the software. For new or modified software instructions for problem reporting should be included in the software user interface as well as in the user's guide.
- development and maintenance of problem reporting webpage(s) for software with unregistered users; the webpage(s) should document reported problems and report the associated corrective actions.

For the modification of existing software or the development of new software, the problem reporting and correction work activity should also include

- a documented mechanism for reporting problems and their disposition within the software development team during software design, implementation, and testing phases of software development
- development and implementation of a method for capturing contact information from registered software users in order to provide prompt problem and correction notification to key members of the user community.

4.10 Training of Personnel

Summary: Appropriate training of software development and maintenance team members ensures that that they are able to perform their design, development, testing, and usage roles in compliance with all applicable technical and SQA requirements.

¹⁵ ASME NQA-1-2000, op. cit., Part IV, Subpart 4.1, Section 204, p. 229.

Priority Rating: Low.

Training project personnel in designing, developing, testing, evaluating, or using software is a way to minimize the likelihood and consequences of software failure. Although software developers usually come into a software project with considerable knowledge and expertise in software development, testing, and project SQA, it is sometimes helpful to provide refresher or customized training to ensure familiarity with all applicable technical and SQA requirements. Training should be commensurate with the scope, complexity, and importance of the tasks and the education, experience, and proficiency of the individual. The organization for which software development team members work should ensure that personnel participate in continuing education and training as necessary to improve their performance and proficiency and ensure that they stay up-to-date on changing technology and new requirements.¹⁶

Although user training is the responsibility of the software user and not the developer, it is the responsibility of the software development team to provide appropriate documentation, user's guide, operator's guides, and testing materials, to enable the user to implement and follow an effective self-training program.

At a minimum, training requirements for software development and maintenance team should be established. Records should be kept on training activities, and training gaps should be identified and communicated to individual project staff members and their management.

¹⁶ DOE-STD-1172-2003, *Safety Software Quality Assurance Functional Area Qualification Standard*, dated 12-03.

5. GRADED APPROACH

The SCAPA approach to SQA guidance relies, as does the DOE SQA approach for safety software, on a graded approach. This approach presents a minimum baseline of SQA for all software projects and applies additional SQA based on the nature of the modifications and intended application of the software.

5.1 Minimum Requirements

For the development of new software products or the modification of existing software (including legacy software), projects are required to meet the minimum SQA standards specified in this guidance document for each of the ten SQA work activities. For the modification of legacy software, there is no requirement to perform retroactive SQA for software management, development, and V&V activities that were conducted when the software was first developed. However, all software modification work on legacy software (e.g., upgrades, error correction) must be conducted using the same SQA guidance and requirements that are specified for the development or modification of non-legacy software.

For the continued use of legacy software, when no additional development or modification is being conducted, minimum SQA standards must be met for work activities that are applicable for the operations, maintenance, and retirement phases of the software life cycle. This means that minimum SQA levels must be met for the following work activities:

- Software Project Management and Quality Planning
- Software Configuration Management.
- Software Design and Implementation
- Verification and Validation
- Problem Reporting and Corrective Action
- Training of Personnel

5.2 Exceeding the Minimum Requirements

Many consequence assessment software products deserve or require additional SQA to meet the needs of the intended application, organizational requirements, or client expectations. When allocating limited resources to focus on additional SQA, Figure 5.1 provides a recommended weighting of the ten categories of SQA work activities for consequence assessment software.

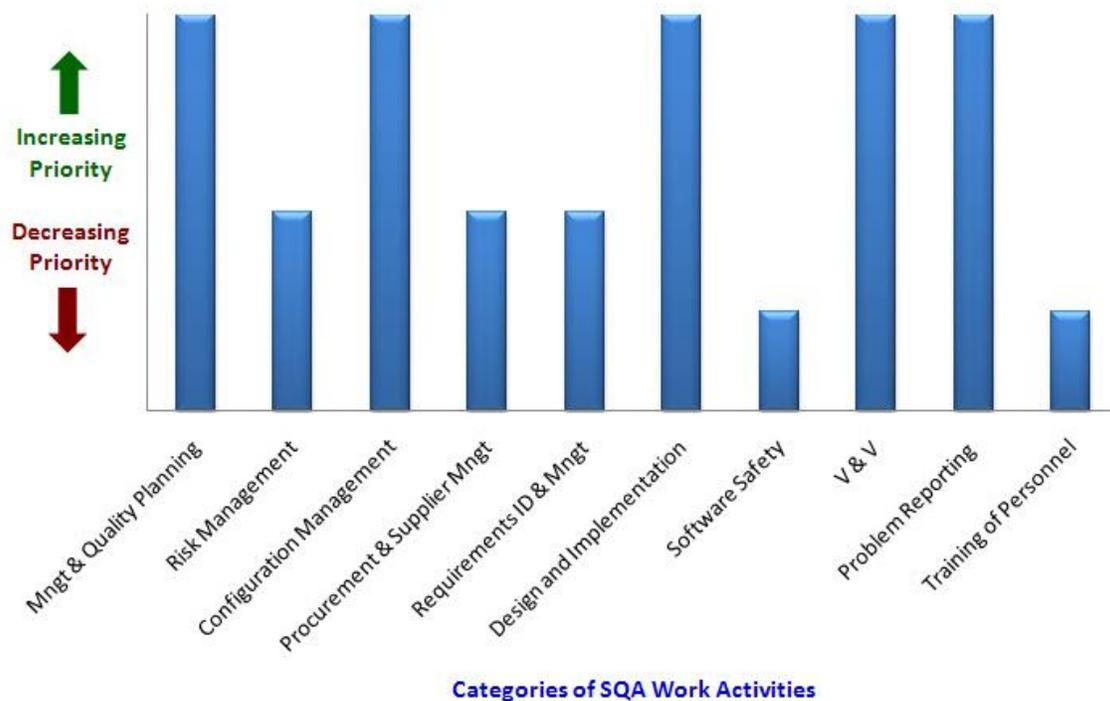


Figure 5.1. Grading the SQA Work Activities

In this approach, the following categories of SQA work activities are given the highest weighting:

- Software Project Management and Quality Planning
- Software Configuration Management
- Software Design and Implementation
- Verification and Validation
- Problem Reporting and Corrective Action.

These are the SQA work activities that often provide the greatest benefit for a given investment of project resources. Within each of these five work activities, there may be grading to suit the size and scope of the software.

The following three SQA work activities are assigned a medium level of priority:

- Software Risk Management
- Procurement and Supplier Management
- Software Requirements Identification and Management.

The larger the software development project and the more staff members involved in the work, the greater the need to perform software risk management and software requirements identification and management activities. Any consequence assessment

software that makes significant use of software products developed outside of the program (with the exception of well-established tools like text editors or compilers) should explicitly consider software procurement and supplier management issues.

The following two work activities are of lower priority:

- Software Safety
- Training of Personnel.

The lower priority rating is a relative score. It does not imply that these work activities are unimportant, only that they have a relatively lower priority when compared to other SQA work activities. The lower priority score was deemed warranted because software safety and training of personnel are often implicitly addressed in other SQA work activities. For large or sensitive projects, these SQA work activities often should be explicitly addressed in appropriate detail to flesh out a comprehensive SQA program.

As discussed throughout this document, it is imperative to achieve an acceptable balance between the need for modeling complex environmental processes, timely innovation, and SQA for safety-related and other non-safety consequences assessment software. Regardless of this balance, the minimum requirements specified for each of the ten SQA work elements should be achieved. If not, departures from this minimum standard should be clearly documented, justified, and approved by the software project SQA representative. Additional SQA activities beyond the minimum expectations should be incorporated as indicated by the needs of the project and the identified balance between SQA and other project priorities. Typically, SQA requirements beyond the minimum specified levels will be directed first toward the high-priority SQA work activities identified above and then to address medium- and lower-priority activities as available resources permit.

GLOSSARY

Acceptance Testing. The process of exercising or evaluating a system or system component by manual or automated means to ensure that it satisfies the specified requirements and to identify differences between expected and actual results in the operating environment. Source: ASME NQA-1-2000.

Administrative Controls. The provisions relating to organization and management, procedures, record keeping, assessment, and reporting necessary to ensure safe operation of a facility. Source: 10 CFR 830.

Assessment. A review, evaluation, inspection, test, check, surveillance, or audit, to determine and document whether items, processes, systems, or services meet specified requirements and perform effectively. Source: DOE Order 414.1C.

Configuration Management. The process of identifying and defining the configuration items in a system (i.e., software and hardware), controlling the release and change of these items throughout the system's life cycle, and recording and reporting the status of configuration items and change requests. Source: ASME NQA-1-2000.

Consequence. An outcome of an event, hazard, threat, or situation. Source: IEEE Std 1540-2001.

Firmware. The combination of a hardware device and computer instructions and data that reside as read-only software on that device. Notes: (1) This term is sometimes used to refer only to the hardware device or only to the computer instructions or data, but these meanings are deprecated. (2) The confusion surrounding this term has led some to suggest that it be avoided altogether. Source: IEEE Std 610.12-1990.

Functional Configuration Audit. An audit conducted to verify that the development of a configuration item has been completed satisfactorily, that the item has achieved the performance and functional characteristics specified in the functional or allocated configuration identification, and that its operational and support documents are complete and satisfactory. Source: IEEE Std-610.12-1990.

Graded Approach. The process of ensuring that the level of analyses, documentation, and actions used to comply with requirements is commensurate with

- the relative importance to safety, safeguards, and security;
- the magnitude of any hazard involved;
- the life-cycle stage of a facility or item;
- the programmatic mission of a facility;
- the particular characteristics of a facility or item;

- the relative importance to radiological and nonradiological hazards; and
- any other relevant factors.

Source: 10 CFR 830.

Hazard Controls. Measures to eliminate, limit, or mitigate hazards to workers, the public, or the environment, including

1. physical, design, structural, and engineering features;
2. safety structures, systems and components;
3. safety management programs;
4. technical safety requirements; and
5. other controls necessary to provide adequate protection from hazards.

Source: 10 CFR 830.

Non-Safety Software. Software that performs functions, or whose output is used to make decisions, that would not have an impact on public or worker safety.

Nuclear Facility. A reactor or a nonreactor nuclear facility where an activity is conducted for or on behalf of DOE and includes any related area, structure, facility, or activity to the extent necessary to ensure proper implementation of the requirements established in 10 CFR 830.

Process. A series of actions that achieves an end result. Source: 10 CFR 830.

Quality. The condition achieved when an item, service, or process meets or exceeds the user's requirements and expectations. Source: 10 CFR 830.

Quality Assurance. All those actions that provide confidence that quality is achieved. Source: 10 CFR 830.

Quality Assurance Program. The overall program or management system established to assign responsibilities and authorities, define policies and requirements, and provide for the performance and assessment of work. Source: 10 CFR 830.

Risk. The likelihood of an event, hazard, threat, or situation occurring and its undesirable consequences; a potential problem. Source: IEEE Standard 1540-2001.

Safety. An all-inclusive term used synonymously with environment, safety, and health to encompass protection of the public, the workers, and the environment. Source: DOE Order 414.1C.

Safety Software. Includes safety system software, safety and hazard analysis software and design software and safety management and administrative controls software (DOE Order 414.1C). *Safety system software* is software for a nuclear facility that performs a safety function as part of an SSC and is cited in either (1) a DOE-approved documented safety analysis or (2) an approved hazard analysis per DOE P 450.4, *Safety Management*

System Policy, dated 10-15-96, and the DEAR clause. *Safety and hazard analysis software and design software* is software that is used to classify, design, or analyze nuclear facilities. This software is not part of an SSC but helps to ensure the proper accident or hazards analysis of nuclear facilities or an SSC that performs a safety function. *Safety management and administrative controls software* is software that performs a hazard control function in support of nuclear facility or radiological safety management programs or technical safety requirements or other software that performs a control function necessary to provide adequate protection from nuclear facility or radiological hazards. This software supports eliminating, limiting, or mitigating nuclear hazards to workers, the public, or the environment, as addressed in 10 CFR 830, 10 CFR 835, and the DEAR ISMS clause.

Safety Structures, Systems, and Components (SSC). Both safety class structures, systems, and components and safety significant structures, systems, and components (10 CFR 830).

Safety-Class Structures, Systems, and Components (SC SSCs). Structures, systems, or components, including portions of process systems, whose preventive and mitigative function is necessary to limit radioactive hazardous material exposure to the public, as determined from the safety analyses. Source: 10 CFR 830.

Safety-Related Software. Software that performs functions, or whose output is used to make decisions, that are associated with an aspect of public or worker safety, but is outside the DOE O 414.1c definition of safety software.

Safety-Significant Structures, Systems, and Components (SS SSCs). Structures, systems, and components which are not designated as safety-class SSCs, but whose preventive or mitigative function is a major contributor to defense in depth and/or worker safety as determined from safety analyses [10 CFR 830]. As a general rule of thumb, safety-significant SSC designations based on worker safety are limited to those systems, structures, or components whose failure is estimated to result in a prompt worker fatality or serious injuries (e.g., loss of eye, loss of limb) or significant radiological or chemical exposure to workers. Source: DOE Guide 420.1-1.

Safety System Software. Software for a nuclear facility¹⁷ that performs a safety function as part of a structure, system or component and is cited in either DOE-approved documented safety analysis or an approved hazard analysis per DOE P 450.4, *Safety Management System Policy*, dated 10-15-96, and the DEAR clause. Source: DOE Order 414.1C.

Service. Work, such as design, construction, fabrication, decontamination, environmental remediation, waste management, laboratory sample analysis, safety software development/validation/testing, inspection, nondestructive examination/testing, environmental qualification, equipment qualification, training, assessment, repair, and installation or the like. Source: 10 CFR 830.

¹⁷ Per 10 CFR 830, quality assurance requirements apply to all DOE nuclear facilities including radiological facilities (see 10 CFR 830, DOE Std 1120, and the DEAR clause).

Software. Computer programs, procedures, and associated documentation and data pertaining to the operation of a computer system. Source: NQA-1-2000.

Verification and Validation. The process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements. Source: IEEE Std-610.12-1990.

6. REFERENCES

10 CFR 830, Nuclear Safety Management.

10 CFR 835, Occupational Radiation Protection.

ASME NQA-1-2000. *Quality Assurance Requirements for Nuclear Facility Applications*. American Society of Mechanical Engineers, New York, 2001.

DNFSB Technical Report 25. *Quality Assurance for Safety-Related Software at Department Of Energy Defense Nuclear Facilities*. Defense Nuclear Facilities Safety Board, January 2000.

DNFSB Recommendation 2002-1. *Implementation Plan for Defense Nuclear Facilities Safety Board Recommendation 2002-1, Quality Assurance for Safety Software at Department of Energy Defense Nuclear Facilities*. Defense Nuclear Facilities Safety Board, September 2002.

DOE Order 414.1C, *Quality Assurance*. U.S. Department of Energy, Washington, D.C., June 2007. Available at <http://www.directives.doe.gov/pdfs/doe/doetext/neword/414/o4141c.pdf>.

DOE Guide 414.1-4. *Safety Software Guide for Use with 10 CFR 830 Subpart A, Quality Assurance Requirements, And DOE Order 414.1C, Quality Assurance*. U.S. Department of Energy, Washington, D.C., June 2005. Available at <http://www.directives.doe.gov/pdfs/doe/doetext/neword/414/g4141-4.pdf>.

DOE Guide 420.1-1. *Nonreactor Nuclear Safety Design Criteria and Explosives Safety Criteria Guide for use with DOE O 420.1, Facility Safety*. U.S. Department of Energy, Washington, D.C., March 2000. Available at www.directives.doe.gov/pdfs/doe/doetext/neword/420/g4201-1.pdf.

DOE P 450.4. *Safety Management System Policy*. U.S. Department of Energy, Washington, D.C., October 1996. Available at <http://www.directives.doe.gov/pdfs/doe/doetext/neword/450/p4501.html>.

DOE-STD-1120-2005. *Integration of Environment, Safety, and Health into Facility Disposition Activities*. U.S. Department of Energy, Washington, D.C., April 2005. Available at www.hss.energy.gov/.../ns/techstds/.../std1120/DOE-STD-1120-1-2005.pdf.

IEEE Std 16085. *IEEE Standard for Software Engineering: Software Life Cycle Processes—Risk Management*. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey, 2004.

IEEE Std 610.12-1990. *IEEE Standard Glossary of Software Engineering Terminology*. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey, 2003.

IEEE Std 1540-2001. *IEEE Standard for Software Life Cycle Processes—Risk Management*. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey, 2001.