Table of Contents

LIST OF FIGURES ...................................................................................................................... ii
LIST OF TABLES ........................................................................................................................ ii
ACRONYMS ................................................................................................................................ iii
PURPOSE ...................................................................................................................................... 1
PREFACE ...................................................................................................................................... 1
REQUIRED TECHNICAL COMPETENCIES ........................................................................ 3
1. Safety software quality assurance personnel must demonstrate a working level knowledge of DOE quality assurance policy, programs, and processes contained in: ............. 3
2. Safety software quality assurance personnel must demonstrate a working level knowledge of the elements of a successful software quality assurance program. .................... 6
3. Safety software quality assurance personnel must demonstrate a working level knowledge of the types of safety software and grading levels. .............................................. 16
4. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the functional interfaces between safety system software and the system-level design. ............................................................................................................................ 20
5. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the safety and engineering scenarios that are modeled by software. ................ 23
6. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the relationships between nuclear hazards and the control and protection functions being addressed by safety management and administrative controls software... 26
7. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the purpose, features, and contents of the DOE safety software Central Registry. .................................................................................................................................. 29
8. Safety software quality assurance personnel must demonstrate a working level knowledge of the safety software life cycle processes. .......................................................... 32
9. Safety software quality assurance personnel must demonstrate a working level knowledge of software requirements identification and management. ............................... 52
10. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the safety software design concepts. .................................................. 59
11. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the safety software design and implementation practices. ................... 65
12. Safety software quality assurance personnel must demonstrate a working level knowledge of the safety software verification and validation processes that ensure software will adequately fulfill all intended safety functions. ........................................ 73
13. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of software safety analysis. ................................................................. 86
14. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of activities that ensure that safety software will be properly maintained and will continue to operate as intended................................................................. 89
15. Safety software quality assurance personnel must demonstrate a working level knowledge of software configuration management.......................................................... 94
16. Safety software quality assurance personnel must demonstrate the ability to perform an assessment of safety software quality using appropriate DOE directives and standards and industry standards................................................................................................. 102

Selected Bibliography and Suggested Reading ........................................................................................................ 109

Tables

Table 1. Example of an RTM................................................................................................................................. 26
Table 2. Mapping safety software types and grading levels to SQA work activities ........................................ 42
Table 3. Sample RTM......................................................................................................................................... 57
Table 4. Measures for initial implementation ................................................................................................... 92
<table>
<thead>
<tr>
<th>ACRONYMS</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASME</td>
<td>American Society of Mechanical Engineers</td>
</tr>
<tr>
<td>COTS</td>
<td>commercial off-the-shelf</td>
</tr>
<tr>
<td>CSCI</td>
<td>computer software configuration items</td>
</tr>
<tr>
<td>CSU</td>
<td>computer software unit</td>
</tr>
<tr>
<td>DFD</td>
<td>data flow diagram</td>
</tr>
<tr>
<td>DoD</td>
<td>U.S. Department of Defense</td>
</tr>
<tr>
<td>DOE</td>
<td>U.S. Department of Energy</td>
</tr>
<tr>
<td>ES&amp;H</td>
<td>environment, safety, and health</td>
</tr>
<tr>
<td>ESTSC</td>
<td>Energy Science and Technology Software Center</td>
</tr>
<tr>
<td>FAQS</td>
<td>functional area qualification standard</td>
</tr>
<tr>
<td>FCA</td>
<td>functional configuration audit</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>KSA</td>
<td>knowledge, skill, and ability</td>
</tr>
<tr>
<td>MEL</td>
<td>Maya embedded language</td>
</tr>
<tr>
<td>M&amp;O</td>
<td>management and operating</td>
</tr>
<tr>
<td>NNSA</td>
<td>National Nuclear Security Administration</td>
</tr>
<tr>
<td>NQA</td>
<td>Nuclear Quality Assurance</td>
</tr>
<tr>
<td>PCA</td>
<td>physical configuration audit</td>
</tr>
<tr>
<td>QA</td>
<td>quality assurance</td>
</tr>
<tr>
<td>QAP</td>
<td>quality assurance program</td>
</tr>
<tr>
<td>RTM</td>
<td>requirements traceability matrix</td>
</tr>
<tr>
<td>RTOS</td>
<td>real time operating system</td>
</tr>
<tr>
<td>SCM</td>
<td>software configuration management</td>
</tr>
<tr>
<td>SDP</td>
<td>software development plan</td>
</tr>
<tr>
<td>SLC</td>
<td>software life cycle</td>
</tr>
<tr>
<td>SLCM</td>
<td>software life cycle model</td>
</tr>
<tr>
<td>SLCP</td>
<td>software life cycle process</td>
</tr>
<tr>
<td>SPMP</td>
<td>software project management plan</td>
</tr>
<tr>
<td>SQA</td>
<td>software quality assurance</td>
</tr>
<tr>
<td>SQAP</td>
<td>software quality assurance plan</td>
</tr>
<tr>
<td>SRS</td>
<td>software requirements specification</td>
</tr>
<tr>
<td>SSC</td>
<td>structure, system, or component</td>
</tr>
<tr>
<td>SSQA</td>
<td>Safety software quality assurance</td>
</tr>
<tr>
<td>SSR</td>
<td>software safety requirement</td>
</tr>
<tr>
<td>ACRONYMS</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>SwSE</td>
<td>software safety engineer</td>
</tr>
<tr>
<td>TQP</td>
<td>Technical Qualification Program</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>verification and validation</td>
</tr>
</tbody>
</table>
PURPOSE

The purpose of this reference guide is to provide a document that contains the information required for a Department of Energy (DOE)/National Nuclear Security Administration (NNSA) technical employee to successfully complete the Safety Software Quality Assurance Functional Area Qualification Standard (FAQS). Information essential to meeting the qualification requirements is provided; however, some competency statements require extensive knowledge or skill development. Reproducing all the required information for those statements in this document is not practical. In those instances, references are included to guide the candidate to additional resources.

SCOPE

This reference guide has been developed to address the competency statements in the (March 2011) edition of DOE-STD-1172-2011, Safety Software Quality Assurance Functional Area Qualification Standard. The qualification standard for safety software quality assurance contains 16 competency statements.

Please direct your questions or comments related to this document to the Learning and Career Development Department.

PREFACE

Competency statements and supporting knowledge and/or skill statements from the qualification standard are shown in contrasting bold type, while the corresponding information associated with each statement is provided below it.

A comprehensive list of acronyms is provided at the beginning of this document. It is recommended that the candidate review the list prior to proceeding with the competencies, as the acronyms, abbreviations, and symbols may not be further defined within the text unless special emphasis is required.

The competencies and supporting knowledge, skill, and ability (KSA) statements are taken directly from the FAQS. Most corrections to spelling, punctuation, and grammar have been made without remark.

Every effort has been made to provide the most current information and references available as of March 2011. However, the candidate is advised to verify the applicability of the information provided. It is recognized that some personnel may oversee facilities that utilize predecessor documents to those identified. In those cases, such documents should be included in local qualification standards via the Technical Qualification Program (TQP).

In the cases where information about an FAQS topic in a competency or KSA statement is not available in the newest edition of a standard (consensus or industry), an older version is referenced. These references are noted in the text and in the bibliography.
Only significant corrections to errors in the technical content of the discussion text source material are identified. Editorial changes that do not affect the technical content (e.g., grammatical or spelling corrections, and changes to style) appear without remark.
REQUIRED TECHNICAL COMPETENCIES

1. Safety software quality assurance personnel must demonstrate a working level knowledge of DOE quality assurance policy, programs, and processes contained in:
   - DOE O 414.1C, *Quality Assurance*
   - 10 CFR 830 Subpart A, “Quality Assurance”

   a. Discuss the purpose and interrelationships of DOE O 414.1C and 10 CFR 830, Subpart A.

The following is taken from DOE O 414.1C.

Safety software quality requirements are necessary to ensure that DOE/NNSA safety software in nuclear facilities performs its intended specific safety functions in relation to structures, systems, or components (SSCs) and that the classification, design, and analysis associated with nuclear facility operations is correct. These requirements complement those of 10 CFR 830, “Nuclear Safety Management,” and provide detail for work associated with safety software that is conducted under the nuclear facility quality assurance program (QAP) compliant with 10 CFR 830.

Federal personnel with software quality assurance (SQA) responsibilities must have technical competency to carry out their duties. Technical qualification requirements will be specified in technical qualification standards. This process is coordinated with the Federal Technical Capability Panel (FTCP).

The Assistant Secretary for Environment, Safety, and Health (ES&H) has lead responsibility for promulgating requirements and guidance for safety software through the directives system after formal coordination with the Office of the Chief Information Officer. ES&H is also responsible for managing the safety software central registry for the Department.

Work processes involving safety software must be developed and implemented using national or international consensus standards and must include the following elements:

- Facility design authority involvement in the identification of software requirements specification (SRS), acquisition, design, development, verification and validation (V&V), including inspection and testing, configuration management, maintenance, and retirement.
- Identify, document, and maintain safety software inventory.
- Establish grading levels for safety software. Document those grading levels in the QAP.
- Using the grading levels established and approved, select and implement applicable SQA work activities to ensure that safety software performs its intended functions. American Society of Mechanical Engineers Nuclear Quality Assurance, ASME NQA-1-2008, *Quality Assurance Requirements for Nuclear Facility Applications*, or other national or international consensus standards that provide an equivalent level of (QA) quality assurance requirements as ASME NQA-1-2008, must be used to
implement these work activities. The standards used must be specified by the user and approved by DOE.

b. **Discuss the DOE and contractor requirements and responsibilities for development, review, approval, and implementation of a quality assurance program (QAP).**

The following is taken from 10 CFR 830.

Contractors conducting activities, including providing items or services, that affect, or may affect, the nuclear safety of DOE nuclear facilities must conduct work in accordance with the QA criteria in 10 CFR 830.122, “Quality Assurance Criteria.”

The contractor responsible for a DOE nuclear facility must
- submit a QAP to DOE for approval and regard the QAP as approved 90 days after submittal, unless it is approved or rejected by DOE at an earlier date.
- modify the QAP as directed by DOE.
- annually submit any changes to the DOE-approved QAP to DOE for approval. Justify in the submittal why the changes continue to satisfy the QA requirements.
- conduct work in accordance with the QAP.

The QAP must
- describe how the QA criteria of 10 CFR. 830.122 are satisfied;
- integrate the QA criteria with the safety management system, or describe how the QA criteria apply to the safety management system;
- use voluntary consensus standards in its development and implementation, where practicable and consistent with contractual and regulatory requirements, and identify the standards used; and
- describe how the contractor responsible for the nuclear facility ensures that subcontractors and suppliers satisfy the criteria of 10 CFR 830.122.

c. **Discuss and give an example of how the safety software inventory requirements specified in DOE O 414.1C can be met.**

The following is taken from DOE Software Quality Assurance Criteria Review and Approach Documents.

The criteria for a safety software inventory include the following:
- Safety software, as identified in DOE O 414.1C, is identified for each DOE nuclear facility.
- An inventory list of the safety software is documented.
- The software inventory includes all versions of specific applications in use at the publication date of the inventory list, and includes software that has been removed and designated not to be used.
d. Discuss the purpose, benefits, and restrictions of the graded approach in the implementation of DOE quality assurance requirements.

The following is taken from DOE O 414.1C.

The process of ensuring that the levels of analyses, documentation, and actions used to comply with requirements are 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.

e. Discuss the implementation of a software quality assurance program such as described in DOE G 414-1-4, Safety Software Guide for use with 10 CFR 830, Subpart A, Quality Assurance Requirements, and DOE O 414.1C, Quality Assurance.

The following is taken from DOE G 414.1-4.

An SQA program establishes the appropriate safety software life-cycle practices, including safety design concepts, to ensure that software functions reliably and correctly performs the intended work specified for that safety software. In other words, SQA’s role is to minimize or prevent the failure of safety software and any undesired consequences of that failure. The rigor imposed by SQA is driven by the intended use of the safety software. More importantly, the rigor of SQA should address the risk of use of such software. Effective safety software quality is one method for avoiding, minimizing, or mitigating the risk associated with the use of the software.

f. Discuss and give examples of how national or international consensus standards can be used in the development and implementation of work processes to provide an equivalent level of quality assurance requirements.

The following is taken from DOE G 414.1-4.

Title 10 CFR 830 Subpart A and DOE O 414.1C require the use of standards to develop and implement a QAP. National/international standards facilitate a common software quality engineering approach to developing or documenting software based upon a consensus of experts in the particular topic areas. Many national and international standards bodies have developed software standards to ensure that the recognized needs of their industry and users are satisfactorily met.

In the United States, ASME is the nationally accredited body for the development of nuclear facility QA standards. DOE O 414.1C cites ASME NQA-1-2008 or other national or
international consensus standards that provide an equivalent level of QA requirements as ASME NQA-1-2008 as the appropriate standard for QAPs applied to nuclear-related activities. The ten QA criteria in 10 CFR 830 Subpart A and DOE O 414.1C are mapped to ASME NQA-1-2008 in appendix C. DOE O 414.1C also requires that additional standards be used to address specific work activities conducted under the QAP, such as safety software.

In the case of ASME NQA-1-2008, part I, the requirements generally apply to safety software work activities. For example, requirements 3, 4, 7, 11, 16, and 17 for design control, procurement document control, control of purchased items and services, test control, corrective action, and QA records (respectively) can have specific safety software applicability. In addition, ASME NQA-1-2008, part II, subpart 2.7, and part IV, subpart 4.1, specifically address “quality assurance requirements for computer software for nuclear facility applications” and “guide on quality assurance requirements for software” (respectively). As stated in the introduction of part II, subpart 2.7, this subpart “provides requirements for the acquisition, development, operation, maintenance, and retirement of software.”

2. **Safety software quality assurance personnel must demonstrate a working level knowledge of the elements of a successful software quality assurance program.**

   a. **Discuss the purpose, scope and content of the following types of software management plan(s) as they relate to safety software quality:**

      - Software project management plan
      - Software risk management plan
      - Software development plan
      - Software safety plan
      - Software quality assurance plan
      - Software test plan
      - Software verification and validation plan
      - Software configuration management plan
      - Software procurement and supplier management plan
      - Software problem reporting and corrective action plan
      - Software integration plan
      - Software maintenance plan
      - Software installation plan
      - Software operations plan
      - Software training plan
      - Software retirement plan

*Software Project Management Plan, Software Development Plan, and Software Quality Assurance Plan*

The following is taken from DOE G 414.1-4.
As with any system, project management and quality planning are key elements to establishing the foundation to ensure a quality product that meets project goals. For software, project management starts with the system level project management and quality planning. Software specific tasks should be identified and either included within the overall system planning or in separate software planning documents.

These tasks may be documented in a software project management plan (SPMP), a software quality assurance plan (SQAP), a software development plan (SDP), or similar documents. They also may be embedded in the overall system. Typically the SPMP, SQAP, and/or SDP are the controlling documents that define and guide the processes necessary to satisfy project requirements, including the software quality requirements. These plans are initiated early in the project life cycle and are maintained throughout the life of the project.

The software project management and quality planning should include identifying all tasks associated with the software development and procurement, including procurement of services, estimate of the duration of the tasks, resources allocated to the task, and any dependencies. The planning should include a description of the tasks and any relevant information.

Software quality and software development planning identifies and guides the software phases and any grading of the SQAP and software development activities performed during software development or maintenance. The software quality and software engineering activities and rigor of implementation will be dependent on the identified grading level of safety software and the ability of DOE or its contractors to build quality in and assess the quality of the safety software. Because SQAP and SDP are overall basic quality and software engineering plans, some quality activities, such as software configuration management (SCM), risk management, problem reporting and corrective actions, and V&V, including software reviews and testing, may be further detailed in separate plans.

Software Risk Management Plan
The following is taken from DOE G 414.1-4.

Software risk management provides a disciplined environment for proactive decision-making to continuously assess what can go wrong, determine what risks are important to address, and implement actions to address those risks. Because risk management is such a fundamental tool for project management, it is an integral part of software project management. Although sometimes associated with safety analysis of potential failures, software risk management focuses on the risks to the successful completion of the software project. The risks addressed by this work activity are project management risks associated with the successful completion of a software application.

Software Safety Plan
The following is taken from National Aeronautics and Space Administration, NASA-STD-8719.13B.
The purpose of a software safety plan is to provide requirements to implement a systematic approach to software safety as an integral part of the project’s overall system safety program, software development, and software assurance processes. It describes the activities necessary to ensure that safety is designed into software that is acquired or developed and that safety is maintained throughout the software and system life cycle. How these requirements are met will vary with the program, project, facility, and mission.

Software Test Plan
The following is taken from DOE G 200.1-1, chapter 4.

The software test plan is a narrative and tabular description of the test activities planned for the project. The software test plan should establish the testing necessary to validate that the project requirements have been met and that the deliverables are at an acceptable level in accordance with existing standards. The plan also ensures that a systematic approach to testing is established and that the testing is adequate to verify the functionality of the software product.

The software test plan includes the resources, project team responsibilities, and management techniques needed to plan, develop, and implement the testing activities that will occur throughout the life cycle. If individuals outside of the project team perform system and acceptance testing, the plan includes the responsibilities and relationships of external test groups.

In this stage, the plan is written at a high level and focuses on identifying test methodologies and test phases. Detailed information about test products is added to the software test plan as the project progresses through subsequent life cycle stages.

Development of the software test plan is the responsibility of the project manager. If a test group outside the project team will be involved in any test phase, the project manager must coordinate the software test plan with each test group.

Software Verification and Validation Plan
The following is taken from DOE G 414.1-4.

Verification and validation is the largest area within the SQA activities. Verification is performed throughout the life cycle of the safety software. Validation activities are performed at the end of the software development or acquisition processes to ensure the software meets the intended requirements. Verification and validation activities should be performed by competent staff other than those who developed the item being verified or validated. Verification and validation activities include reviews, inspections, assessments, observations, and testing.

Reviews and inspections of software deliverables requirement specifications, procurement documents, software design, code modules, test results, training materials, user
documentation, and processes that guide the software development activities should be performed.

Software Configuration Management Plan
The following is taken from DOE G 414.1-4.

Software configuration management activities identify all functions and tasks required to manage the configuration of the software system, including software engineering items, establishing the configuration baselines to be controlled, and software configuration change control process. The following four areas of SCM should be addressed when performing configuration management:

1. Configuration identification
2. Configuration control
3. Configuration status accounting
4. Configuration audits and reviews

Software Procurement and Supplier Management Plan
The following is taken from DOE G 414.1-4.

Most software projects will have procurement activities that require interactions with suppliers regardless of whether the software is level A, B, or C. Procurement activities may be as basic as the purchase of compilers or other development tools for custom-developed software or as complicated as procuring a complete safety system software control system. Thus, there are a variety of approaches for software procurement and supplier management based on

- the level of control DOE or its contractors have on the quality of the software or software service being procured, and
- the complexity of the software.

Procurement documentation should include the technical and quality requirements for the safety software. Some of the specifications that should be included are

- specifications for the software features, including requirements for safety, security, functions, and performance;
- process steps used in developing and validating the software, including any documentation to be delivered;
- requirements for supplier notification of defects, new releases, or other issues that impact the operation; and
- mechanisms for the users of the software to report defects and request assistance in operating the software.

These requirements should be assessed for completeness and to ensure the quality of the software being purchased. There are four major approaches for this assessment:

1. Performing an assessment of the supplier
2. Requiring the supplier to provide a self-declaration that the safety software meets the intended quality
3. Accepting the safety software based on key characteristics
4. Verifying the supplier has obtained a certification or accreditation of the software product quality or software quality program from a third party

**Software Problem Reporting and Corrective Action Plan**
The following is taken from DOE G 414.1-4.

Coupled with the configuration management of the software system, the problem reporting and corrective action process should address the appropriate requirements of the QAP corrective action system. The reporting and corrective action system will cover 1) methods for documenting, evaluating and correcting software problems; 2) an evaluation process for determining whether a reported problem is indeed a defect or an error; and 3) the roles and responsibilities for disposition of the problem reports, including notification to the originator of the results of the evaluation. If the noted problem is indeed an error, the problem reporting and corrective action system should correlate the error with the appropriate software engineering elements; identify the potential impacts and risks to past, present, and future developmental and operational activities; and support the development of mitigation strategies. After an error has been noted, all users should be apprised to ascertain any impacts upon safety basis decisions.

Procurement documents should identify the requirements for suppliers to report problems to the supplier, any required supplier response, and the method for the purchasers to report problems to the supplier.

Maintaining a robust problem reporting and corrective action process is obviously vital to maintaining a reliable and vital safety software system. 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.

This work activity should be fully implemented for all level A and B software types and for level C custom-developed. This formal implementation should include documentation and tracking to closure of any problems reported for the software and authorization to perform the corrective action. A graded approach that reduces the formality of documenting problem reports and approving corrective actions taken may be applied for level A and B utility calculation safety software and all level C software applications except custom-developed. This less formal implementation may include interoffice communications describing the problem identified and the corrective actions planned.

**Software Integration Plan**
The following is taken from DOE G 200.1-1, chapter 8.

Software integration and testing activities focus on interfaces between and among components of the software product, such as functional correctness, system stability, overall system operability, system security, and system performance requirements. Software
integration and testing performed incrementally provides feedback on quality, errors, and design weaknesses early in the integration process.

In this stage, software components are integrated and tested to determine whether the software product meets predetermined functionality, performance, quality, interface, and security requirements. Once the software product is fully integrated, system testing is conducted to validate that the software product will operate in its intended environment, satisfies all user requirements, and is supported with complete and accurate operating documentation.

*Software Maintenance Plan*

The following is taken from DOE G 200.1-1, chapter 10.

The basic maintenance process model includes input, process, output, and control for software maintenance. It is based on the same software engineering principles and preferred practices that lower risk and improve quality during the planning and development stages of the life cycle. The process model supports the concept that planned software changes should be grouped and packaged into scheduled releases that can be managed as projects. This proven approach allows the maintenance team to better plan, optimize use of resources, take advantage of economies of scale, and better control outcome in terms of both schedule and product quality.

*Software Installation Plan*

The following is taken from DOE G 200.1-1, chapter 9.

Installation and acceptance of the software product are initiated after the system test has been successfully completed. This stage involves the activities required to install the software, data bases, or data that comprise the software product onto the hardware platform at the site(s) of operation. The objectives of the activities in this stage are to verify that the software product meets design requirements and to obtain the system owner’s acceptance and approval of the software product. The activities associated with this stage should be performed each time the software product is installed at an acceptance test site or production site.

*Software Operations Plan*

The following is taken from DOE G 200.1-1, appendix D.

The continuity of operations plan or statement is reviewed to determine if it includes the following elements:

- Requirements for continuity of operation, such as data backup, data recovery, and operation start-up
- Plans for backup and recovery operations
- Training requirements and plans so that the required skills will be in place to execute backup and recovery operations
Software Training Plan

The following is taken from DOE G 200.1-1, appendix D.

The software training plan is reviewed to determine if it includes the following elements in the level of detail needed for training staff to begin logistical training arrangements:

- **Background**—system description, objectives, curriculum overview
- **Training requirements**—environment, audience, category, skill level of users
- **Objective**—expected results of the training in terms of the increased level of user knowledge
- **Training strategy**—type of training, schedule, duration, sites
- **Training resources**—resources required, responsibilities of involved parties
- **Environment**—facilities, support from other groups, equipment, actions required
- **Training materials**—types of materials required

Software Retirement Plan

The following is taken from DOE G 200.1-1, appendix F.

The software retirement plan establishes the process for the orderly retirement of information systems regardless of software platform or size, both classified and unclassified. It also addresses the disposition of information systems records according to DOE requirements. It may not include the removal of information systems from the software libraries used for software reuse and sharing. The software retirement plan should be followed to remove software, data, and references to an information system from all computer operating platforms and to notify other offices that keep records of computer applications that a system has been retired.

b. **Identify and describe safety software procurement methods, including supplier evaluation and source inspection processes.**

The following is taken from DOE G 414.1-4.

Most software projects will have procurement activities that require interactions with suppliers regardless of whether the software is level A, B, or C. Procurement activities may be as basic as the purchase of compilers or other development tools for custom-developed software or as complicated as procuring a complete safety system software control system. There are a variety of approaches for software procurement and supplier management based on

- the level of control DOE or its contractors have on the quality of the software or software service being procured
- the complexity of the software

Procurement documentation should include the technical and quality requirements for the safety software. Some of the specifications that should be included are

- specifications for the software features, including requirements for safety, security, functions, and performance;
- process steps used in developing and validating the software, including any documentation to be delivered;
- requirements for supplier notification of defects, new releases, or other issues that impact the operations; and mechanisms for the users of the software to report defects and request assistance in operating the software.

The requirements should be assessed for completeness and to ensure the quality of the software being purchased. There are four major approaches for this assessment:

1. Performing an assessment of the supplier
2. Requiring the supplier to provide a self-declaration that the safety software meets the intended quality
3. Accepting the safety software based on key characteristics
4. Verifying the supplier has obtained a certification or accreditation of the software product quality or software quality program from a third party

c. Using the references below, describe the various elements of an acceptable safety software quality assurance program for the development, use, grading, and maintenance of safety software.

- 10 CFR 830, “Nuclear Safety Management”
- DOE O 414.1C, Quality Assurance
- DOE G 414.1-4, Safety Software Guide for use with 10 CFR 830 Subpart A, Quality Assurance Requirements, and DOE O 414.1C, Quality Assurance
- DOE O 200.1A, Information Management Program
- ASME NQA-1-2000, Quality Assurance Program Requirements for Nuclear Facilities Applications, including Part I, Part II, Subpart 2.7, and Part IV, Subpart 4.1

10 CFR 830, Nuclear Safety Management

Where appropriate, a contractor must use a graded approach to implement the requirements of 10 CFR 830, document the basis of the graded approach used, and submit that documentation to DOE.

The QAP must address the management, performance, and assessment criteria described in 10 CFR 830.122.

DOE O 414.1C, Quality Assurance

Safety software quality requirements are necessary to ensure that DOE/NNSA safety software in nuclear facilities performs its intended specific safety functions in relation to SSCs and that the classification, design, and analysis associated with nuclear facility operations is correct. These requirements complement those of 10 CFR 830, and provide detail for work associated with safety software that is conducted under the nuclear facility QAP compliant with 10 CFR 830.
Federal personnel with SQA responsibilities must have technical competency to carry out their duties. Technical qualification requirements will be specified in technical qualification standards. This process is coordinated with the FTCP.

The Assistant Secretary for Environment, Safety, and Health has lead responsibility for promulgating requirements and guidance for safety software through the directives system after formal coordination with the Office of the Chief Information Officer. ES&H is also responsible for managing the safety software central registry for the Department.

Work processes involving safety software must be developed and implemented using national or international consensus standards and must include the following elements:

- Facility design authority involvement in the identification of SRS, acquisition, design, development, V&V, including inspection and testing, configuration management, maintenance, and retirement.
- Identify, document, and maintain safety software inventory.
- Establish grading levels for safety software. Document those grading levels in the QAP.
- Using the grading levels established and approved, select and implement applicable SQA work activities to ensure that safety software performs its intended functions. ASME NQA-1-2000, or other national or international consensus standards that provide an equivalent level of quality assurance requirements as ASME NQA-1-2000, must be used to implement these work activities. The standards used must be specified by the user and approved by DOE.

DOE G 414.1-4, Safety Software Guide for Use with 10 CFR 830 Subpart A, Quality Assurance Requirements, and DOE O 414.1C, Quality Assurance

It is important that SQA is part of an overall QAP required for nuclear facilities in accordance with 10 CFR 830, and DOE O 414.1C. Regardless of the application of the software, an appropriate level of quality infrastructure should be established and a commitment made to maintain this infrastructure for the safety software.

An SQA program establishes the appropriate safety software life cycle practices, including safety design concepts, to ensure that software functions reliably and correctly performs the intended work specified for that safety software. In other words, SQA’s role is to minimize or prevent the failure of safety software and any undesired consequences of that failure. The rigor imposed by SQA is driven by the intended use of the safety software. More importantly, the rigor of SQA should address the risk of use of such software. Effective safety software quality is one method for avoiding, minimizing, or mitigating the risk associated with the use of the software.

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. SQA cannot address the risks created by the failure of other system components but can address the software reaction to effects caused by these
types of failures. SQA should not be isolated from system level quality assurance (QA) and other system level activities. In many instances, hardware fail-safe methods are implemented to mitigate risk of safety software failure. Additionally other interfaces such as hardware and human interfaces with safety software should implement QA activities.

**DOE O 200.1A, Information Management Program**

DOE management must ensure that DOE’s acquisition, use, and management of IT hardware and software meet program and mission goals as well as promote sound resource management and a more cost-effective government, in accordance with Federal Acquisition Regulation Part 39, Acquisition of Information Technology, and DOE O 413.3A, *Program and Project Management for the Acquisition of Capital Assets*; specifically to

- promote consolidation of software package acquisition, volume purchasing arrangements, enterprise-wide agreements and best practices in software implementation, consistent with the SmartBuy program, identified in OMB Memo (M-03-14), Reducing Cost and Improving Quality in Federal Purchases of Commercial Software;
- ensure that all departmental software engineering uses a risk and life cycle based SQA program, in compliance with DOE N 203.1, *Software Quality Assurance* and DOE O 414.1C; and
- in accordance with Federal Acquisition Regulation
  - establish centralized authorities to coordinate hardware and software purchases,
  - deploy acquisition strategies for IT hardware and software, designed to take advantage of volume discount savings,
  - promote use of common hardware and software configurations, where appropriate, and
  - adopt standard replacement policies to make the best use of existing resources.

**ASME NQA-1-2000, Quality Assurance Requirements for Nuclear Facility Applications**

ASME NQA-1-2000 provides requirements and guidelines for the establishment and execution of QAPs during siting, design, construction, operation, and decommissioning of nuclear facilities. NQA-1-2000 reflects industry experience and current understanding of the QA requirements necessary to achieve safe, reliable, and efficient utilization of nuclear energy, and management and processing of radioactive materials. NQA-1-2008 focuses on the achievement of results, emphasizes the role of the individual and line management in the achievement of quality, and fosters the application of these requirements in a manner consistent with the relative importance of the item or activity.

**IEEE 730.1, IEEE Standard for Software Quality Assurance Plans**

The purpose of IEEE 730.1 is to provide uniform, minimum acceptable requirements for preparation and content of SQA plans. IEEE 730.1 applies to the development and maintenance of critical software. For non-critical software, or for software already developed, a subset of the requirements of this standard may be applied. An assessment should be made for the specific software item to ensure adequacy of coverage. Where this
standard is invoked for an organization or project engaged in producing several software items, the applicability of the standard should be specified for each of the software items.

Development Plans
The software development plan (SDP) can be used as the highest-level planning document governing a project, or could be subordinate within a larger set of plans. For example, several SDPs may be written in support of a larger project that is governed by an SPMP. The SDP should identify all technical and managerial activities associated with software development. The SDP should specify the following items, which should be reviewed and assessed by the SQA organizational element:

- Description of software development
- Software development organization responsibilities and interfaces
- Process for managing the software development
- Technical methods, tools, and techniques to be used in support of the software development

Maintenance
A maintenance manual should contain instructions for software product support and maintenance, such as procedures for correcting defects, installation of enhancements, and testing of all changes. All hardware and software configuration specifications required to maintain the software should be described in detail.

Any unusual settings or known anomalies should be identified to aid in efficient maintenance. New versions of software should be thoroughly tested prior to incorporation into operational systems. Version control procedures should be reviewed and approved by SQA and SCM organizational elements. The SQA organizational element should periodically audit and validate the use of the version control procedures as well as the software maintenance process and procedures.

3. Safety software quality assurance personnel must demonstrate a working level knowledge of the types of safety software and grading levels.

   a. Explain the general characteristics, application, and safety significance of safety software including: safety system software; safety and hazard analysis software and design software; safety management and administration controls software.

Safety System Software
The following is taken from DOE G 414.1-4.

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, and the Department of Energy Acquisition Regulations clause.
**Safety and Hazard Analysis and Design Software**

The following is taken from DOE O 414.1C.

Safety and hazard analysis and design software 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 Administration Controls Software**

The following is taken from DOE O 414.1C.

Safety management and administration controls software 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.

**b. Given examples of safety software, identify the grading levels and the applicable SQA work activities associated with the safety software (such as defined in DOE G 414.1-4 Safety Software Guide for use with 10 CFR 830 Subpart A, Quality Assurance Requirements, and DOE O 414.1C, Quality Assurance).**

This is a performance-based KSA. The Qualifying Official will evaluate its completion. The following information taken from DOE G 414.1-4 may be helpful.

Proper implementation of DOE O 414.1C will be enhanced by grading safety software based on its application. Safety software grading levels should be described in terms of safety consequence and regulatory compliance. The grading levels are defined as follows.

**Level A**

This grading level includes safety software applications that meet one or more of the following criteria:

- Software failure that could compromise a limiting condition for operation
- Software failure that could cause a reduction in the safety margin for a safety SSC that is cited in DOE-approved documented safety analysis
- Software failure that could cause a reduction in the safety margin for other systems such as toxic or chemical protection systems that are cited in either a DOE-approved documented safety analysis or an approved hazard analysis
- Software failure that could result in non-conservative safety analysis, design, or misclassification of facilities or SSCs

**Level B**

This grading level includes safety software applications that do not meet level A criteria but meet one or more of the following criteria:
• Safety management databases used to aid in decision-making whose failure could impact safety SSC operation
• Software failure that could result in incorrect analysis, design, monitoring, alarming, or recording of hazardous exposures to workers or the public
• Software failure that could comprise the defense-in-depth capability for the nuclear facility

**Level C**
This grading level includes software applications that do not meet level B criteria but meet one or more of the following criteria:
• Software failure that could cause a potential violation of regulatory permitting requirements
• Software failure that could affect ES&H monitoring or alarming systems
• Software failure that could affect the safe operation of an SSC

c. **Describe the process for identifying safety software.**

The following is taken from DOE, Pacific Northwest National Laboratory, New DOE Software Quality Assurance Requirements and Implications for Meteorology and Consequence Assessment Software, What is Safety Software?

Safety software is
• software that performs a safety function as part of an SSC at a nuclear facility and is cited in either 1) a DOE-approved documented safety analysis or 2) an approved hazard analysis.
• 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.
• software that meets one or more of the following:
  o performs a hazard control function in support of nuclear facility or radiological safety management program
  o performs a control function necessary to provide adequate protection from nuclear facility or radiological hazards
  o supports eliminating, limiting, or mitigating nuclear hazards to workers, the public, or the environment

d. **Describe the function of the following safety software types (such as defined in DOE G 414.1-4) and applications and provide an example of each:**
• Custom-developed
• Configurable
• Acquired
• Utility calculations
• Commercial design and analysis

The following descriptions and examples are taken from DOE G 414-1.4.
**Custom-developed**

Custom-developed software is built specifically for a DOE application or to support the same function for a related government organization. It may be developed by DOE or one of its management and operating (M&O) contractors or contracted with a qualified software company through the procurement process. Examples of custom-developed software includes material inventory and tracking database applications, accident consequence applications, control system applications, and embedded custom-developed software that controls a hardware device.

**Configurable**

Configurable software 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. An example is software associated with programmable logic controllers.

**Acquired**

Acquired software is generally supplied through basic procurements, two-party agreements, or other contractual arrangements. Acquired software includes commercial off-the-shelf (COTS) software, such as operating systems, database management systems, compilers, software development tools, and commercial calculational software and spreadsheet tools. Downloadable software that is available at no cost to the user is also considered acquired software. Firmware is acquired software. Firmware is usually provided by a hardware supplier through the procurement process and cannot be modified after receipt.

**Utility Calculations**

Utility calculation software typically uses COTS spreadsheet applications as a foundation and user-developed algorithms or data structures to create simple software products. The utility calculation software is used frequently to perform calculations associated with the design of an SSC. Utility software that is used with high frequency may be labeled as custom software and may justify the same SSQA work activities as custom-developed software. With utility calculation software, it is important to recognize the difference between QA of the algorithms, macros, and logic that perform the calculations versus QA of the COTS software itself. Utility calculation software includes the associated data sets, configuration information, and test cases for validation and/or calibration.

**Commercial Design and Analysis**

Commercial design and analysis software is used in conjunction with design and analysis services provided to DOE from a commercial contractor. An example would be where DOE or an M&O contractor contracts for specified design services support. The design service provider uses its independently developed or acquired software without DOE involvement or support. DOE then receives a completed design. Procurement contracts can be enhanced to require that the software used in the design or analysis services meet the requirements in DOE O 414.1C.
e. Differentiate between real time and non-real time software.

The following is taken from Answers.com.

The key difference between general-computing operating systems and real-time operating systems (RTOSs) is the need for deterministic timing behavior in the RTOS. Deterministic timing means that operating system services consume only known and expected amounts of time. In theory, these service times could be expressed as mathematical formulas. These formulas must be strictly algebraic and not include any random timing components. Random elements in service times could cause random delays in application software and could then make the application randomly miss real-time deadlines. Many non-real-time operating systems also provide similar kernel services.

General-computing non-RTOS are often quite non-deterministic. Their services can inject random delays into application software and thus cause slow responsiveness of an application at unexpected times. If you ask the developer of a non-real-time operating system for the algebraic formula describing the timing behavior of one of its services (such as sending a message from task to task), you will invariably not get an algebraic formula. Deterministic timing behavior is not a design goal for these general-computing operating systems.

Many RTOS proponents argue that a RTOS must not use virtual memory concepts, because paging mechanics prevent a deterministic response. While this is a frequently supported argument, it should be noted that the term RTOS and determinism in this context covers a very wide meaning, and vendors of many different operating systems apply these terms with varied meaning. When selecting an operating system for a specific task, the real-time attribute alone is an insufficient criterion, therefore. Deterministic behavior and deterministic latencies have value only if the response lies within the boundaries of the physics of the process that is to be controlled. For example, controlling a combustion engine in a racing car has different real-time requirements to the problem of filling a one million-liter water tank through a 2-inch pipe.

Real-time operating systems are often used in embedded solutions, that is, computing platforms that are within another device. Examples of embedded systems include combustion engine controllers or washing machine controllers and many others. Desktop PC and other general-purpose computers are not embedded systems. While RTOS are typically designed for and used with embedded systems, the two aspects are essentially distinct, and have different requirements. A RTOS for embedded system addresses both sets of requirements.

4. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the functional interfaces between safety system software and the system-level design.

a. Identify how safety system-level requirements are established and then assigned to hardware, software, and human components.

The following is taken from DOE G 414.1-4.
Proper implementation of DOE O 414.1C will be enhanced by grading safety software based on its application. Safety software grading levels should be described in terms of safety consequence and regulatory compliance. The grading levels are defined as follows.

**Level A**
This grading level includes safety software applications that meet one or more of the following criteria:
- Software failure that could compromise a limiting condition for operation
- Software failure that could cause a reduction in the safety margin for a safety SSC that is cited in DOE-approved documented safety analysis
- Software failure that could cause a reduction in the safety margin for other systems such as toxic or chemical protection systems that are cited in either a DOE-approved documented safety analysis or an approved hazard analysis
- Software failure that could result in non-conservative safety analysis, design, or misclassification of facilities or SSCs

**Level B**
This grading level includes safety software applications that do not meet level A criteria but meet one or more of the following criteria:
- Safety management databases used to aid in decision-making whose failure could impact safety SSC operation
- Software failure that could result in incorrect analysis, design, monitoring, alarming, or recording of hazardous exposures to workers or the public
- Software failure that could comprise the defense-in-depth capability for the nuclear facility

**Level C**
This grading level includes software applications that do not meet level B criteria but meet one or more of the following criteria.
- Software failure that could cause a potential violation of regulatory permitting requirements
- Software failure that could affect ES&H monitoring or alarming systems
- Software failure that could affect the safe operation of an SSC

b. **Identify the typical requirements that define functional interfaces between safety system software components and the system-level design, as described in standards such as IEEE 830, *IEEE Recommended Practice for Software Requirements Specifications* and IEEE 7-4.3.2, *Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations*.**

The following is taken from the Institute of Electrical and Electronic Engineers, IEEE 830, *Recommended Practice for Software Requirements Specifications*. 
The SRS should list each system interface and identify the functionality of the software to accomplish the system requirement and the interface description to match the system.

User Interfaces
User interfaces should specify the following:
- The logical characteristics of each interface between the software product and its users. This includes those configuration characteristics necessary to accomplish the software requirements.
- All the aspects of optimizing the interface with the person who must use the system. This may simply comprise a list of do’s and don’ts on how the system will appear to the user. One example may be a requirement for the option of long- or short-error messages. Like all others, these requirements should be verifiable.

Hardware interfaces
Hardware interfaces should specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics. It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. For example, terminal support may specify full-screen support as opposed to line-by-line support.

Software Interfaces
Software interfaces should specify the use of other required software products, and interfaces with other application systems. For each required software product, the following should be provided:
- Name
- Mnemonic
- Specification number
- Version number
- Source

The following should be provided for each interface:
- Discussion of the purpose of the interfacing software as related to this software product.
- Definition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document defining the interface is required.

c. Explain with an example how the functional interfaces between components and the system-level design are established and how the safety system software controls the safety functions of the subsystems, components, and interfaces.

The following is taken from DOE G 200.1-1, chapter 4.
Functional requirements define what the software product must do to support the system owner’s business functions and objectives. The functional requirements should answer the following questions:

- How are inputs transformed into outputs?
- Who initiates and receives specific information?
- What information must be available for each function to be performed?

Identify requirements for all functions whether they are to be automated or manual. Describe the automated and manual inputs, processing, outputs, and conditions for all functions. Include a description of the standard data tables and data or records that will be shared with other applications. Identify the forms, reports, source documents, and inputs/outputs that the software product will process or produce to help define the functional requirements.

A functional model should be developed to depict each process that needs to be included. The goal of the functional model is to represent a complete top-down picture of the software product.

Flow diagrams should be used to provide a hierarchical and sequential view of the system owner’s business functions and the flow of information through the processes.

5. **Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the safety and engineering scenarios that are modeled by software.**

   a. Explain with an example the sequence of steps that are typically followed in the development process of a safety analysis or design code such as the mathematical model and the associated computational model.

The following is taken from DOE G 414.1-4.

Software rarely functions as an independent entity. Software is typically a component of a system much in the same way hardware, data, and procedures are system components. Therefore, to understand the risk associated with the use of software, the software function should be considered a part of an overall system function.

The consequences of software faults need to be addressed in terms of the contribution of a software failure to an overall system failure. Issues such as security, training of operational personnel, electromagnetic interference, human factors, or system reliability have the potential to be safety issues. For example, if the security of the system can be compromised, then the safety software can also be compromised. Controlling access to the system is important to maintaining the integrity of the safety software. Likewise, if human factor issues such as ambient lighting conditions and ease of use for understandability are important, the risks need to be addressed either via design or training. For programmable logic controllers or network safety software applications, electromagnetic interference could offer potential risks for operation of the safety software system.
Once the software’s function within the overall system’s function is known, the appropriate software life cycle (SLC) and system life cycle practices can be identified to minimize the risk of software failure on the system. Rigor can then be applied commensurate with the risk. Managing the risk appropriately is the key to managing a safety software system. Unless risks and trade-offs of either doing or not doing an activity are evaluated, there is not a true understanding of the issues involved regarding the safety software system. Obviously, time and resource constraints should be balanced with the probability of occurrence and the potential consequences versus an occurrence of the worst case scenario. If the safety systems staff zealously and religiously inappropriately invokes the strictest rigor for a Level B application, then the application has the potential never to get fielded properly. On the other hand, if the process activities are only minimally or inappropriately performed for a level A software safety application, then very adverse consequences could potentially occur for which no mitigation strategy exists. Appropriate project management is a risk management strategy and especially so for safety software applications.

Toolbox codes represent a small number of standard computer models or codes supporting DOE safety analysis. These codes have widespread use and are of appropriate qualification for use within DOE. The toolbox codes are acknowledged as part of DOE’s safety software central registry. These codes are verified and validated and constitute a safe harbor methodology. That is to say, the analysts using these codes do not need to present additional defense as to their qualification provided that the analysts are sufficiently qualified to use the codes and the input parameters are valid. These codes may also include commercial or proprietary design codes where DOE considers additional SQA controls are appropriate for repetitive use in safety applications and there is a benefit to maintain centralized control of the codes. The following six widely applied safety analysis computer codes have been designated as toolbox codes:

1. ALOHA (chemical dispersion analysis)
2. CFAST (fire analysis)
3. EPIcode (chemical dispersion analysis)
4. GENII (radiological dispersion analysis)
5. MACCS2 (radiological dispersion analysis)
6. MELCOR (leak path factor analysis)

b. Identify and discuss with an example safety analysis scenarios appropriate to simulate with safety software.

This is a performance-based KSA. The Qualifying Official will evaluate its completion.

c. Identify and discuss with an example appropriate uses of design software to assist with safety-related design decisions.

This is a performance-based KSA. The Qualifying Official will evaluate its completion. The following information taken from Joint Services Computer Resources Management Group, Software System Safety Handbook, may be helpful.
One of the primary analyses performed during detailed design analysis is to identify the computer software unit (CSU) where a software safety requirement (SSR) is implemented. The term CSU refers to the code-level routine, function, or module. The best way to accomplish this task is for the software safety engineer (SwSE) to sit down with the software developer, V&V engineer, or QA engineer and to begin to tag/link individual SSRs to CSUs, as illustrated in figure 1.

This accomplishes two goals. Firstly, it helps focus the SwSE on the safety-related processing, which is more important on large-scale development projects than on smaller, less complex programs. Secondly, it provides an opportunity to continue development of the requirements traceability matrix (RTM).

The RTM contains multiple columns, with the left-most column containing the list of SSRs. Adjacent to this column is a description of the SSR, and the name of the computer software configuration item (CSCI) where the individual SSR has been implemented. Adjacent to this column is a column containing the name of the CSU where the SSR has been implemented. The next column contains the name of the test procedure that verifies the implementation of the SSR, and the last column documents test results with pertinent comments. Table 1
illustrates one example of an RTM. Another example is illustrated in table 3 of this reference guide.

In some cases, it may only be possible to tag/link the SSR to the CSCI level due to the algorithms employed or the implementation of the SSR. If this is the case, the SSR will probably not be verified through analysis, but by an extensive testing effort. The RTM should also be included as part of the safety requirements criteria analysis report to provide the evidence that all of the safety requirements have been identified and traced to the design and test.

Table 1. Example of an RTM

<table>
<thead>
<tr>
<th>SSR</th>
<th>Requirement Description</th>
<th>CSCI</th>
<th>CSU</th>
<th>Test Procedure</th>
<th>Test Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Source: Joint Services Computer Resources Management Group, Software System Safety Handbook*

Once the RTM has been populated and all SSRs have been tagged/linked to the application code, it is time to start analyzing the individual CSUs to ensure that the intent of the SSR has been satisfied. Again, in order to accomplish this task, it is best to have the appropriate developers and/or engineers available for consultation. Process flow charts and data flow diagrams (DFDs) are excellent examples of soft tools that can aid in this process. These tools can help the engineer review and analyze software safety interlocks such as checks, flags, and firewalls that have been implemented in the design. Process flows and DFDs are also useful in performing “What If” types of analyses, performing safety critical path analyses, and identifying potential hazards related to interfacing systems. The software analysis folder should include the products resulting from these safety tasks.

6. **Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the relationships between nuclear hazards and the control and protection functions being addressed by safety management and administrative controls software.**

   a. **Identify how the software functional requirements of safety management and administrative controls software are defined, documented, and controlled relative to nuclear hazard controls and protection, risk management, and design constraints.**

   The following is taken from DOE G 200.1-1, chapter 4.
Define Functional Requirements

Functional requirements define what the software product must do to support the system owner’s business functions and objectives. The functional requirements should answer the following questions:

- How are inputs transformed into outputs?
- Who initiates and receives specific information?
- What information must be available for each function to be performed?

Identify requirements for all functions whether they are to be automated or manual. Describe the automated and manual inputs, processing, outputs, and conditions for all functions. Include a description of the standard data tables and data or records that will be shared with other applications. Identify the forms, reports, source documents, and inputs/outputs that the software product will process or produce to help define the functional requirements.

A functional model should be developed to depict each process that needs to be included. The goal of the functional model is to represent a complete top-down picture of the software product. Flow diagrams should be used to provide a hierarchical and sequential view of the system owner’s business functions and the flow of information through the processes.

Define Input and Output Requirements

Describe all manual and automated input requirements for the software product such as data entry from source documents and data extracts from other applications.

Describe all output requirements for the software product such as printed reports, display screens, and files.

Define Performance Requirements

Performance requirements define how the software product must function. The information gathered in defining the project objectives can translate into very specific performance requirements. Also, government and DOE policy can dictate specific availability and response times.

Define User Interface Requirements

The user interface requirements should describe how the user will access and interact with the software product, and how information will flow between the user and the software product.

The following are some of the issues that should be considered when trying to identify user interface requirements:

- The users’ requirements for screen elements, navigation, and help information
- The standards for the programmatic organization, DOE, government, and industry that apply to user interfaces
- The range of skill levels of the users who will access and use the software product
- The range of work that the users will be performing with the software product
Define the user interface requirements by identifying and understanding what is most important to the user, not what is most convenient for the project team. Work with the system owner and users to develop a set of user interface requirements that can be used for all automated products for the system owner’s organization. A standard set of user interface requirements will simplify the design and code processes, and ensure that all automated products have a similar look and feel to the users. When other constraints do not permit the use of existing user interface standards, an attempt should be made to keep the user interface requirements as close as possible to the existing standard.

**Define System Interface Requirements**

The hardware and software interface requirements must specify hardware and software interfaces required to support the development, operation, and maintenance of the software product. The following information should be considered when defining the hardware and software interface requirements:

- System owner’s and users’ computing environment
- Existing or planned software that will provide data to or accept data from the software product
- Other organizations or users having access to the software product
- Purpose or mission of interfacing software
- Common users, data elements, reports, and sources for forms/events/outputs
- Timing considerations that will influence sharing of data, direction of data exchange, and security constraints
- Development constraints such as the operating system, database management system, language compiler, tools, utilities, and network protocol drivers
- Standardized system architecture defined by hardware and software configurations for organizations, programmatic offices, or telecommunications programs

**Define Communication Requirements**

The communication requirements define connectivity and access requirements within and between user locations and between other groups and applications. The following factors should be considered when defining communication requirements:

- Communication needs of the user and customer organizations
- User organization’s existing and planned telecommunications environment
- Projected changes to the current communication architecture, such as the connection of additional local and remote sites
- Limitations placed on communications by existing hardware and software, including
  - existing user systems
  - existing applications that will interface with the software product
  - existing organizations that will interface with the software product
- Organization, government, and industry standards that define communication requirements and limitations
Define Computer Security and Access Requirements

Develop the computer security requirements in conjunction with the system owner’s computer system security officer or the assistant computer protection program manager. This involvement affords early determination of classifications and levels of protection required for the software product.

If a software product under development processes sensitive personal information, appropriate safeguards must be established to protect the information from accidental disclosure.

Implement applicable security procedures to ensure data integrity and protection from unauthorized disclosure, particularly during development efforts. The organization that owns the data defines the data classification. The project team must be aware of all the types of data and of any classified or proprietary algorithms used in the software product.

Define Backup and Recovery Requirements

Develop the requirements for data backup, recovery, and operation startup for the software product in conjunction with the site authority for continuity of operations. If a software product has been defined as mission essential, a continuity of operations plan must be developed.

b. Given the scope of safety management and administrative controls software, explain how the radiological hazard controls and protection being addressed by the software were translated into software functional requirements, and how the software supports nuclear hazards risk management functions.

This is a performance-based KSA. The Qualifying Official will evaluate its completion.

7. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the purpose, features, and contents of the DOE safety software Central Registry.

a. Discuss the purpose of the safety software Central Registry and identify the current codes in the Central Registry.

The following is taken from DOE, Office of Health, Safety, and Security, Safety Software Quality Assurance—Central Registry.

A significant improvement in the safety software used at the DOE nuclear facilities was achieved with the establishment of a list of toolbox codes that are compliant with the DOE SSQA requirements of DOE O 414.1C and its guidance, DOE G 414.1-4. The toolbox codes are routinely used by DOE to perform calculations and to develop data used to establish the safety basis for DOE facilities and their operation, and to support the variety of safety analyses and safety evaluations developed for these facilities. The list of toolbox codes is
referred to as the DOE safety software central registry. The following six widely applied safety analysis computer codes have been designated as toolbox codes:

1. ALOHA (chemical dispersion analysis)
2. CFAST (fire analysis)
3. EPIcode (chemical dispersion analysis)
4. GENII (radiological dispersion analysis)
5. MACCS2 (radiological dispersion analysis)
6. MELCOR (leak path factor analysis)

b. Discuss the intent and use of the following documents as they relate to the codes in the safety software Central Registry:

- Code gap analysis report
- Code guidance report

**Code Gap Analysis Report**

The following is taken from DOE, Path Forward to Address Gaps in Toolbox Code Gap Analysis Reports.

Six toolbox codes were added to the safety software central registry as part of the Department’s implementation plan for Defense Nuclear Facilities Safety Board Recommendation 2002-1, *Quality Assurance for Safety Related Software*. The implementation plan required the Department to perform a gap analysis on the six original toolbox codes to determine the actions needed to bring the codes into compliance with SQA criteria. A gap analysis was conducted for each of the six toolbox codes and the actions needed to bring the codes into compliance with SQA criteria have been identified.

The gap analysis was based on a set of SQA requirements and criteria generally compliant with ASME NQA-1-2008. Each toolbox code was evaluated against ten SQA criteria requirements.

The six toolbox codes are considered safety analysis software per the definition in DOE O 4 14.1C.

The toolbox code gap analysis performed in 2004 was conducted using a ten point criteria to define and evaluate the SLC activities related to: 1) software classification; 2) SQA procedures and plans; 3) requirement phase; 4) design phase documentation; 5) implementation phase; 6) testing phase; 7) user instructions; 8) acceptance test; 9) configurations control; and 10) error impact.

The ten-point criteria were not in existence at the time the six toolbox codes were developed and issued for use. As a result, the toolbox codes do not meet many of these system development life cycle criteria. However, NQA-1-2008, subpart 2.7, section 302 provides specific requirements for accepting acquired software that were previously approved under a program which is not consistent with NQA-1-2008. The specific code section states that:
Software that has not been previously approved under a program consistent with this standard for use in its intended application (e.g., freeware, shareware, procured commercial off-the-shelf, or otherwise acquired software) shall be evaluated in accordance with the requirements of this Subpart. The software shall be identified and controlled prior to evaluation. The evaluation, specified by this section, shall be performed and documented to determine adequacy to support operation and maintenance and identify the activities to be performed and the documentation that is needed.

This determination shall be documented and shall identify as a minimum:

- capabilities and limitations for intended use
- test plans and test cases required to demonstrate the capabilities within the limitations
- instructions for use within the limits of the capabilities

Exceptions from the documentation requirements of this subpart and the justification for acceptance shall be documented.

The results of the above evaluation and the performance of the actions necessary to accept the software shall be reviewed and approved. The resulting documentation and associated computer program(s) shall establish the current baseline.

Revisions to previously baseline software received from organizations not required to follow this subpart shall be evaluated in accordance with this section.

The six toolbox codes meet the NQA-1-2008 definition acquired software because the Department does not own the toolbox codes and the toolbox codes were either developed at the direction of and/or with funding from other government agencies or they are privately owned. Using the NQA-1-2008, section 302, subpart 2.7 criteria and mapping them against the ten-point criteria used in the gap analysis, indicates that the section 302 criteria are adequately satisfied by the criteria 6 through 10 of the gap analysis reports.

The emphasis here is on well-documented reference and user manual, code validation, configuration control and error reporting and corrective action.

Further review of the 2004 gap analysis documents revealed that some of the reported deficiencies pertaining to several sub-elements of the criteria 6 through 10 were attributed, for example, to lack of explicit documentation pertaining to criteria 2 through 5, or documentation not available during the gap analysis effort. Of the ten criteria, number 1 is a determination made by the code user and the criteria 2 through 5 are normally performed before the software is issued for general use.

*Code Guidance Report*

The following is taken from DOE-EH-4.2.1.3—ALOHA Code Guidance.
To ensure appropriate application of the designated toolbox software, the implementation plan has committed to sponsoring a set of code-specific documents to guide informed use of the software, and supplement the available user’s manual information.

The ALOHA guidance report includes the following:
- Applicability information for documented safety analysis, specifically tailored for DOE safety analysis
- Code development information and SQA background
- Appropriate regimes and code limitations
- Valid ranges of input parameters consistent with code capability and DOE safety basis applications
- Default input value recommendations for site-independent parameters

**c. Using DOE G 414.1-4, Safety Software Guide for use with 10 CFR 830 Subpart A, Quality Assurance Requirements, and DOE O 414.1C, Quality Assurance, as a reference, describe the SQA requirements that are applicable to the Central Registry codes and the SQA activities the DOE users need to perform for the Central Registry codes.**

The following is taken from DOE G 414.1-4.

In the typical life cycle processes associated with most software applications, updates, improvements, and modifications will be made. Similar to software that is being considered for the first time, revised software in the form of a new software version may also be submitted for inclusion in the central registry, with accompanying removal of the older version.

The steps may be summarized as follows:
- The software sponsor identifies the SQA evaluator organization.
- The evaluator performs a complete evaluation of all aspects of the new software version, emphasizing new and revised aspects of the software application.
- Upon conclusion of the evaluation and issuance of the SQA evaluation report (the gap analysis), the software sponsor decides whether software has satisfactorily met all requisite criteria for the ten SQA work activities. The revised software may be submitted to the central registry.

**8. Safety software quality assurance personnel must demonstrate a working level knowledge of the safety software life cycle processes.**

a. **Discuss the differences between various life-cycle models, such as waterfall, spiral, incremental, and evolutionary, and how they relate to the software development processes.**

The following is taken from U.S. Air Force, Software Technology Support Center, Chapter 2, Software Life Cycle.
Waterfall

The waterfall model, also known as the linear sequential model, is shown in figure 2 with its major phases, milestones, and products. It is a highly structured development process, first used on U.S. Department of Defense (DoD) software projects in the 1970s.

It is the traditional approach to software development and was derived from defense and aerospace project life cycles. It is considered superior to the previously used “code and fix” methods of software development, which lacked formal analysis and design.

Source: U.S. Air Force, Software Technology Support Center

Figure 2. Waterfall model

The waterfall model is documentation-intensive, with earlier phases documenting what must be done and subsequent phases adding greater detail and defining how it should be done. The output from one phase serves as the input to the next phase, with the project flowing from one step to the next in a waterfall fashion. Phases are assumed to be sequential, with only localized feedback during the transition between phases. This is accomplished by using reviews as gates. Comprehensive reviews validate the work of one phase, and require the resolution of any problems before development is allowed to proceed to the next phase.

An important consideration for the waterfall model is that fixes or modifications are often put off until the maintenance phase. This can be very costly, as the cost to correct a problem gets higher with each successive phase.
Advantages

- System is well documented.
- Phases correspond with project management phases.
- Cost and schedule estimates may be lower and more accurate.
- Details can be addressed with more engineering effort if software is large or complex.

Disadvantages

- All risks must be dealt with in a single software development effort.
- Because the model is sequential, there is only local feedback at the transition between phases.
- A working product is not available until late in the project.
- Progress and success are not observable until the later stages. If a mistake or deficiency exists in the documentation of earlier phases, it may not be discovered until the product is delivered.
- Corrections must often wait for the maintenance phase.

Application

The waterfall model can be successfully used when requirements are well understood in the beginning and are not expected to change or evolve over the life of the project. Project risks should be relatively low.

Spiral

The spiral model was developed with the goal of reducing risk in the SLC. It combines elements of the waterfall, evolutionary, and incremental models, and depending on how it is implemented can strongly resemble any combination of the others. The model’s spiral nature can be seen in figure 3, one of several variants. The project starts at the center and progresses through multiple cycles, each working through the software development activities associated with the four quadrants:

1. Determine objectives, alternatives, constraints.
2. Evaluate alternatives. Identify and resolve risks.
3. Develop the next-level product.
4. Plan the next phase.

Risk management is a key element of the spiral model and each round of the spiral identifies problems with the highest risk and develops solutions for that set of problems. The process may even resemble a waterfall with additional risk management techniques. Each cycle ends in a review in which stakeholders agree on plans for the next cycle. The software is usually not developed for release until the last cycle.
The spiral model has been used extensively in the commercial world because of its performance in a market-driven environment. It significantly reduces technical risk and more easily incorporates new technology and innovations. At the same time, it tends to increase cost and schedule risks. In the past the spiral model has been difficult to implement in the DoD environment. This is because contracts with predefined deliverables and schedules do not easily accommodate repeating phases, requirement tradeoffs, and changing deliverables. The spiral model has become somewhat popular in the defense and aerospace industries.

Advantages
- It provides better risk management than other models.
- Requirements are better defined.
- System is more responsive to user needs.

Disadvantages
- The spiral model is more complex and harder to manage.
- This method usually increases development costs and schedule.

Source: U.S. Air Force, Software Technology Support Center

Figure 3. The spiral model
Application
The spiral method should be considered for projects where risks are high, requirements must be refined, and user needs are very important.

Incremental
The incremental model is essentially a series of waterfall cycles. One variant is shown in figure 4. The requirements are known at the beginning of the project and are divided into groups for incremental development. A core set of functions is identified in the first cycle and is built and deployed as the first release. The software development cycle is repeated, with each release adding more functionality until all requirements are met. Each development cycle acts as the maintenance phase for the previous software release. While new requirements that are discovered during the development of a given cycle can be implemented in subsequent cycles, this model assumes that most requirements are known up front. The effort is planned and executed to satisfy the initial list of requirements. A modification to the incremental model allows development cycles to overlap. That is, a subsequent cycle may begin before the previous cycle is complete.

Source: U.S. Air Force, Software Technology Support Center

Figure 4. The incremental model is a series of waterfalls

Advantages
- Provides some feedback, allowing later development cycles to learn from previous cycles.
- Requirements are relatively stable and may be better understood with each increment.
- Allows some requirements modification and may allow the addition of new requirements.
- It is more responsive to user needs than the waterfall model.
- A usable product is available with the first release, and each cycle results in greater functionality.
- The project can be stopped any time after the first cycle and leave a working product.
- Risk is spread out over multiple cycles.
- This method can usually be performed with fewer people than the waterfall model.
- Return on investment is visible earlier in the project.
- Project management may be easier for smaller, incremental projects.
- Testing may be easier on smaller portions of the system.
Disadvantages
- The majority of requirements must be known in the beginning.
- Formal reviews may be more difficult to implement on incremental releases than on a complete system.
- Because development is spread out over multiple iterations, interfaces between modules must be well-defined in the beginning.
- Cost and schedule overruns may result in an unfinished system.
- Operations are impacted as each new release is deployed.
- Users are required to learn how to use a new system with each deployment.

Application
The incremental model is good for projects where requirements are known at the beginning, but need functionality early in the project or that can benefit from the feedback of earlier cycles. Because each cycle produces a working system, it may also be advantageous for projects whose continued funding is not assured and may be cut at any time. It is best used on low to medium-risk programs. If the risks are too high to build a successful system using a single waterfall cycle, spreading the development out over multiple cycles may lower the risks to a more manageable level.

Evolutionary
The evolutionary model, like the incremental model, develops a product in multiple cycles. Unlike the incremental model, which simply adds more functionality with each cycle, this model produces a more refined prototype system with each iteration. The process, shown in figure 5, begins in the center with initial requirements and plans, and progresses through multiple cycles of planning, risk analysis, engineering, and customer evaluation.

![Evolutionary model diagram]

Source: U.S. Air Force, Software Technology Support Center

Figure 5. Evolutionary model

Each cycle produces a prototype that the customer evaluates, followed by a refinement of requirements. Specification, development, and testing activities are carried out concurrently.
(in the engineering quadrant) with rapid feedback. Since requirements continue to change, documentation is minimal, although essential information must still be included for understanding the system and for future support. Implementation compromises are often made in order to get the prototype working—permanent fixes can be made with the next prototype. Operational capability is achieved early, but users must be willing to learn how to use each new prototype.

General system requirements must be known prior to development. This is particularly helpful where evolving technology is being introduced into the project. The evolutionary model relies heavily on user feedback after each implementation to refine requirements for the next evolutionary step.

Advantages
- Project can begin without fully defining or understanding requirements.
- Final requirements are improved and more in line with real user needs.
- Risks are spread over multiple software builds and controlled better.
- Operational capability is achieved earlier in the program.
- Newer technology can be incorporated into the system as it becomes available during later prototypes.
- Documentation emphasizes the final product instead of the evolution of the product.
- This method combines a formal specification with an operational prototype.

Disadvantages
- Because there are more activities and changes, there is usually an increase in both cost and schedule over the waterfall method.
- Management activities are increased.
- Instead of a single switch over to a new system, there is an ongoing impact to current operations.
- Configuration management activities are increased.
- Greater coordination of resources is required.
- Users sometimes mistake a prototype for the final system.
- Prototypes change between cycles, adding a learning curve for developers and users.
- Risks may be increased in the following areas:
  - Requirements—temptation to defer requirements definition.
  - Management—programs are more difficult to control. Better government/contractor cooperation needed.
  - Approval—vulnerable to delays in funding approval, which can increase schedule and costs.
  - Architectural—initial architecture must accommodate later changes.
  - Short-term benefits—risk of becoming driven by operational needs rather than program goals.
  - Risk avoidance—tendency to defer riskier features until later.
  - Exploitation by suppliers—government bargaining power may be reduced because initial contract may not complete the entire task, and subsequent contracts are not likely to be competed.
Patchwork quilt effects—if changes are poorly controlled, the product quality can be compromised.

Application
The evolutionary model can be employed on most types of acquisitions. However, it is usually employed on medium to high-risk systems. The evolutionary model should be considered for systems where requirements are not all known or not yet refined, but are expected to evolve. It is more applicable to new systems than upgrading existing software. The developing and using organizations must be flexible and willing to work with evolving prototypes. Programs well-suited to employ the evolutionary model have some or all of the following characteristics:

- Software intensive systems
- Have a large number of diverse users
- Have rapidly changing software technology
- Are developing an unprecedented system
- Humans are an integral part of the system
- Limited capability is needed quickly

b. Describe each phase, associated SQA work activities, and products of a typical software life-cycle model such as the one described in DOE G 414.1-4, Safety Software Guide for use with 10 CFR 830 Subpart A, Quality Assurance Requirements, and DOE O 414.1C, Quality Assurance and IEEE 1074, IEEE Standard for Developing Software Project Life Cycle Processes. Explain the roles of quality assurance, configuration management, and verification and validation in each phase.

The following is taken from the Institute of Electrical and Electronics Engineers, IEEE 1074.

IEEE 1074, Standard for Developing Software Life Cycle Processes, presents a description of the way in which software implementation is to be approached.

The process architect has primary responsibility for creating and maintaining the SLC plan. This responsibility is implemented in three steps: 1) select a software life cycle model (SLCM), 2) create an SLC, and 3) select a software life cycle process (SLCP).

Select a Software Life Cycle Model
Initially, the process architect shall identify the SLCM to which the activities will be mapped. This step encompasses locating, evaluating, selecting, and acquiring an SLCM. It is possible for an organization to have multiple SLCMs; however, only one model is to be selected for a project.

The process architect shall follow the following five steps to evaluate and select an SLCM:
1. Identify all the SLCMs that are available to the development project.
2. Identify the attributes that apply to the desired end system and the development environment.
3. Identify any constraints that might be imposed on the selection.
4. Evaluate the various SLCMs based on past experience and organizational capabilities.
5. Select the SLCM that will best satisfy the project attributes and constraints.

Create an SLC

The activities should be mapped onto the SLCM. The components of mapping are as follows.

Place the Activities in Executable Sequence

The order in which activities will be performed will be determined by three major factors:

1. The selected SLCM will dictate an initial ordering of activities. As mapping progresses, the actual order in which activities will be performed will be established.
2. Schedule constraints may require the overlapping of activities in the SLCM and may thus impact the ordering. In this case, activities may be mapped for parallel execution rather than for serial execution.
3. The ordering of activities may be impacted by the entry and exit criteria of associated activities. The availability of output information from one activity could affect the start of another activity. The second activity may require, as inputs, one or more of the outputs of the first activity. For example, no software design of any kind can be done unless some minimum information is available about software requirements. Another example is that no evaluation activities can be performed unless there is some output product upon which to work.

Develop and Justify a List of Activities Not Used

All “if applicable” activities that do not apply to this project shall be identified and explained in the list of activities not used.

Verify the Map

The process architect shall ensure that the activities are fully mapped onto the selected SLCM and that the resulting SLC contains all of the activities that are necessary to successfully complete a software project. The process architect shall also verify that the information flow into and out of the activities will support the relative order into which they have been mapped.

Establish an SLCP

The preceding steps develop the SLC. As the next step, the available organization process assets shall be applied to the SLC activities, and the known constraints shall be reconciled. The output information that is generated by each activity shall be assigned to the appropriate document(s). The result is the established SLCP.

The following is taken from DOE G 414.1-4.

Software should be controlled in a traceable, planned, and orderly manner. The safety software quality work activities provide the basis for planning, implementing, maintaining, and operating safety software. The work activities for safety software include tasks such as software project planning, SCM, and risk analysis that cross all phases in the life cycle.
Additionally, the work activities include tasks that are specific to a life cycle phase. These work activities cover tasks during the development, maintenance, and operations of safety software. The work activities should be implemented based on the graded level of the safety software and the applicable software type. Table 2 provides a summary of the mapping between software type, the grading levels, and the ten SQA work activities. Not all work activities will be applicable for a particular instance of safety software. The following paragraphs indicate where these work activities may be omitted. However, the best judgment of the software quality engineering and safety system staffs should take precedence over any optional work activities presented in DOE G 414.1-4.

**Software Project Management and Quality Planning**

As with any system, project management and quality planning are key elements to establishing the foundation to ensure a quality product that meets project goals. For software, project management starts with the system level project management and quality planning. Software specific tasks should be identified and either included within the overall system planning or in separate software planning documents.

These tasks may be documented in an SPMP, an SQA plan, an SDP, or similar documents. They also may be embedded in the overall system. Typically the SPMP, SQAP, and/or SDP are the controlling documents that define and guide the processes necessary to satisfy project requirements, including the software quality requirements. These plans are initiated early in the project life cycle and are maintained throughout the life of the project.

The software project management and quality planning should include identifying all tasks associated with the software development and procurement, including procurement of services, estimate of the duration of the tasks, resources allocated to the task, and any dependencies. The planning should include a description of the tasks and any relevant information. In addition to NQA-1-2008, several consensus standards provide details of planning documents that are good resources to assist in the identification and description of the software development and procurement tasks.
Table 2. Mapping safety software types and grading levels to SQA work activities

<table>
<thead>
<tr>
<th>SQA Work Activity</th>
<th>Level A</th>
<th>Level B</th>
<th>Level C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software (S/c) project management &amp; quality planning</td>
<td>Full</td>
<td>Grade</td>
<td>n/a</td>
</tr>
<tr>
<td>Sur risk management</td>
<td>Full</td>
<td>Grade</td>
<td>n/a</td>
</tr>
<tr>
<td>Sur configuration management</td>
<td>Full</td>
<td>Grade</td>
<td>Grade</td>
</tr>
<tr>
<td>Procurement &amp; supplier management</td>
<td>Full</td>
<td>Grade</td>
<td>Grade</td>
</tr>
<tr>
<td>Sur requirements identification &amp; management</td>
<td>Full</td>
<td>Grade</td>
<td>Grade</td>
</tr>
<tr>
<td>Sur design &amp; implementation</td>
<td>Full</td>
<td>n/a</td>
<td>Grade</td>
</tr>
<tr>
<td>Sur safety</td>
<td>Full</td>
<td>Grade</td>
<td>n/a</td>
</tr>
<tr>
<td>Verification &amp; validation</td>
<td>Full</td>
<td>Grade</td>
<td>n/a</td>
</tr>
<tr>
<td>Problem reporting &amp; corrective action</td>
<td>Full</td>
<td>Grade</td>
<td>Grade</td>
</tr>
<tr>
<td>Training of... safety Sw</td>
<td>Full</td>
<td>Grade</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Source: DOE G 414.1-4

Software quality and software development planning identifies and guides the software phases and any grading of the SQA and software development activities performed during software development or maintenance. The software quality and software engineering activities and rigor of implementation will be dependent on the identified grading level of safety software and the ability of DOE or its contractors to build quality in and assess the quality of the safety software. Because SQAPs and SDP plans are overall basic quality and software engineering plans, some quality activities such as SCM, risk management, problem reporting and corrective actions, and V&V, including software reviews and testing, may be further detailed in separate plans.

Software project management and quality planning fully apply to custom-developed and configurable software types for level A and level B safety software. For level A and level B acquired and utility calculation and all level C software applications, software project management and quality planning tasks can be graded. This grading should include the identification and tracking of all significant software tasks. Where instances of the SLC may include little or no software development activities, the software project and quality planning will most likely be part of the overall system level project or facility planning. This work activity does not apply to commercial design and analysis software because the project
management and quality planning activities associated with commercial design and analysis software are performed by the service supplier. DOE controls the SQA activities of that software through procurement agreements and specifications.

Software Risk Management

Software risk management provides a disciplined environment for proactive decision-making to continuously assess what can go wrong, determine what risks are important to address, and implement actions to address those risks. Because risk management is such a fundamental tool for project management, it is an integral part of software project management. Although sometimes associated with safety analysis of potential failures, software risk management focuses on the risks to the successful completion of the software project. The risks addressed by this work activity are project management risks associated with the successful completion of a software application.

Risk assessment and risk control are two fundamental activities required for project success. Risk assessment addresses identification of the potential risks, analysis of those risks, and prioritizing the risks to ensure that the necessary resources will be available to mitigate the risks. Risk control addresses risk tracking and resolution of the risks. Identification, tracking, and management of the risks throughout all phases of the project’s life cycle should include special emphasis on tracking the risks associated with costs, resources, schedules, and technical aspects of the project. Several risk identification techniques are described and detailed in standards and literature.

Risk resolution includes risk avoidance, mitigation, or transference. Even the small risks during one phase of the safety software application’s life have the potential to increase in some other phase of the application’s life with very adverse consequences. In addition, mitigation actions for some risks could create new risks.

The risks associated with the safety software application need to be understood and documented. Each risk should be evaluated against its risk thresholds. Different techniques may be used to evaluate the risks.

Flexibility may need to be applied regarding risk management based on the risk categorization of the safety software application. For a level A safety software, all apparent risks known at the time should be identified, analyzed for impact and probability of occurrence, prioritized, resolved to an acceptable level of risk, and tracked through the life of the safety software. For level B or level C software applications, the granularity for the risks to be identified, analyzed, prioritized, resolved to an acceptable level or risk, and tracked should be determined by the safety system staff and can be graded. The safety system staff should focus on the adverse events that would dominate the risk and assess these in a qualitative manner. The safety system staff also has the responsibility to determine a graded approach for resolving the risks and the process for tracking the risks.
Software Configuration Management

Software configuration management activities identify all functions and tasks required to manage the configuration of the software system, including software engineering items, establishing the configuration baselines to be controlled, and software configuration change control process. The following four areas of SCM should be addressed when performing configuration management:

1. Configuration identification
2. Configuration control
3. Configuration status accounting
4. Configuration audits and reviews

The methods used to control, uniquely identify, describe, and document the configuration of each version or update of software and its related documentation should be documented. This documentation may be included in an SCM plan or its equivalent. Such documentation should include criteria for configuration identification, change control, configuration status accounting, and configuration reviews and audits.

During operations, authorized users lists can be implemented to ensure that the software use is limited to those persons trained and authorized to use the software.

A baseline labeling system should be implemented that uniquely identifies each configuration item, identifies changes to configuration items by revision, and provides the ability to uniquely identify each configuration. This baseline labeling system is used throughout the life of the software development and operation.

Proposed changes to the software should be documented, evaluated, and approved for release. Only approved changes should be made to the software that has been baselined. Software verification activities should be performed for the change to ensure the change was implemented correctly. This verification should also include any changes to the software documentation.

Audits or reviews should be conducted to verify that the software product is consistent with the configuration item descriptions in the requirements and that the software, including all documentation, being delivered is complete. Software configuration management work activities should be applied beginning at the point of DOE’s or its contractor’s control of the software.

For custom-developed safety software graded at level A or level B, all four areas of SCM apply. For all other types of safety software graded as level A or level B and all level C safety software, this work activity may be graded by the optional performance of configuration audits and reviews.

Procurement and Supplier Management

Most software projects will have procurement activities that require interactions with suppliers regardless of whether the software is level A, B, or C. Procurement activities may
be as basic as the purchase of compilers or other development tools for custom-developed software or as complicated as procuring a complete safety systems software control system. Thus, there are a variety of approaches for software procurement and supplier management based on the level of control DOE or its contractors have on the quality of the software or software service being procured and the complexity of the software.

Procurement documentation should include the technical and quality requirements for the safety software. Some of the specifications that should be included are

- specifications for the software features, including requirements of safety, security, functions, and performance;
- process steps used in developing and validating the software, including any documentation to be delivered;
- requirements for supplier notification of defects, new releases, or other issues that impact the operations; and
- mechanisms for the users of the software to report defects and request assistance in operating the software.

These requirements should be assessed for completeness and to ensure the quality of the software being purchased. There are four major approaches for this assessment:

1. Performing an assessment of the supplier
2. Requiring the supplier to provide a self-declaration that the safety software meets the intended quality
3. Accepting the safety software based on key characteristics
4. Verifying the supplier has obtained a certification or accreditation of the software product quality or software quality program from a third party

Software Requirements Identification and Management

Safety system requirements provide the foundation for the requirements to be implemented in the software. These system requirements should be translated into requirements specific for the software. The identified software requirements may be documented in system level requirements documents, SRS, procurement contracts, and/or other acquired software agreements. These requirements should identify functional; performance; security, including user access control; interface and safety requirements; and installation considerations and design constraints where appropriate. The requirements should be complete, correct, consistent, clear, verifiable, and feasible.

User access control during operations is an important aspect to ensuring only authorized users can operate the system or use the software for design or analysis tasks. Controlling access is a software safety and/or security requirement that can be associated with training or qualification to operate the system.

Once the 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 SLC.
This work activity has no grading associated with its performance. Software requirements identification management and traceability applies to level A, B, and C software applications and should fully meet this requirement. However, the detail and format of the safety software requirements may vary with the software type. Custom-developed software most likely will contain a larger number of software requirements than configurable, acquired, utility calculation, or commercial design and analysis tool software, and thus, a separate, more formal document may be applicable.

**Software Design and Implementation**

During software design and implementation the software is developed, documented, reviewed, and controlled. The software design elements should identify the operating system, function, interfaces, performance requirements, installation considerations, design inputs, and design constraints. The software design should be complete and sufficient to meet the software requirements. The design activities and documentation should be adequate to fully describe how the software will interface with other system components and how the software will function internally. Data structure requirements and layouts may be necessary to fully understand the internal operations of the software.

Custom-developed software will require more formality in the documentation and review of the design than configurable or utility calculations. Simple process flows, relationships between data elements, interfaces with external components, and basic database table structures may be all that are needed for configurable or utility calculations, whereas for custom-developed software, complete functional and logical designs of the software components, the input and output data, and pseudo code may be required to fully understand the safety software design. The software design description may be combined with the documentation of the software requirements or software source code.

During implementation, static analysis, clean room inspections, and reviews are common techniques to ensure the implementation remains consistent with the design and does not add complexity or functions that could decrease the safe operation of the software. Many tools exist to evaluate the complexity and other attributes of the source code design structure. Walkthroughs and more formal inspections, such as Fagan inspections, can be used to identify defects in source code, as well as design descriptions and other software development process outputs.

The software developer should perform unit testing prior to system level V&V techniques, including acceptance testing. Developer testing can be very structured and formal, using automated tools or less formal methods. In addition to unit testing, functional, structural, timing, stress, security, and human-factors testing are useful testing methods. These methods can be applied using a graded or tailored approach to ensure the known risks are mitigated appropriately. Other techniques such as error seeding; equivalence class testing; branch and path testing, statistical-based, boundary value testing; and code coverage analysis may all be beneficial testing techniques to ensure robust and reliable software.
The software design and implementation work activity for levels A, B, and C custom-developed software applications should fully meet this requirement. For this software type, the design, including interfaces and data structures, should be completely documented; reviews of the design and code should be performed. Additionally, formal developer testing that includes functional, structural, timing, stress, security, and human-factors testing should be planned, performed and the results documented. It is recommended that the complexity of the custom-developed safety software be evaluated and analysis performed to reduce the complexity of the source code modules.

Configurable and utility calculation for levels A, B, and C software applications may be graded for this work activity. This grading should include fully performing the design work activities as with custom-developed software. However, less formal design and code review, such as simple desk checks by another individual other than that developer, may be performed. Developer testing should be performed and documented that includes safety functions, security, and performance testing. This work activity does not apply to acquired or commercial design and analysis safety software types since the design and implementation activities associated with commercial design and analysis software are performed by the service supplier.

**Software Safety**

The development of software applications requires identification of hazards that have the potential for defeating a safety function and the implementation of design strategies to eliminate or mitigate those hazards. Hence, it is recommended that the software safety process address the mitigation strategy for the components that have potential safety consequences if a fault occurs, where the software design and implementation process addresses the architecture of the safety software application.

Software is only one component of the overall safety system. It may be embedded in an instrumentation and control system, it may be a custom control system for hardware components, or it may be stand alone software used in safety management or support decisions. In any of these or other application to software important to safety, analysis of the software application occurs first at the system level. The analysis should then be performed at the software component level to ensure adequate safeguards are provided to eliminate or mitigate the potential occurrence of a software defect that could cause a system failure.

Methods to mitigate the consequences of software failures should be an integral part of the software design. Specific software analysis and design methods for ensuring that safety functions are well thought out and addressed properly should be performed throughout the software development and operations life cycles. These methods include dynamic and static analyses.

During the initial concept and requirement analysis phases for the software, potential failures need to be identified and evaluated for their consequences of failure and probability of occurrence. Some potential problems are complex or faulty algorithm, lack of proper handing
of incorrect data or error conditions, buffer overflow, and incorrect sequence of operations due to either logic or timing faults.

There are several hazard analysis techniques that may be used for this purpose. Many of these techniques are performed as preliminary analyses and later updated as more information is known about the requirements and design structure. These techniques include failure modes and effects analysis, fault-tree modeling, event-tree modeling, cause-consequence diagrams, hazard and operability analysis, and interface analysis. Techniques such as these should be applied and appropriately documented to understand and assess the impact of software failures on the system.

The design of the software is critical to ensuring safe operation of the system. The software design should consider principles of simplicity, decoupling, and isolation to eliminate the hazards. Complexity of the software design, including the logic and number of data inputs, has proven to increase the defect density in software components. The safety features should be separate from non-safety modules, minimizing the impact of failure of one module on another. The interfaces between the modules need to be defined and tested thoroughly. Separation of the safety features also allows for more rigorous software development and verification practices to be applied to the safety components while providing the appropriate and cost-effective level of SQA applied to the non-safety components. Software engineering safety design practices should include process flow analysis, data flow analysis, path analysis, interface analysis, and interrupt analysis during the design phase.

When hazards related to software functions cannot be eliminated, the hazard should be reduced and/or monitored. Additionally, software can experience partial failures that can degrade the capabilities of the overall system that may not be immediately detectable by the system. In these instances, other design techniques, such as building fault detection and self-diagnostics into the software, should be implemented. Self-diagnostics detect and report software faults and failures in a timely manner and allow actions to be taken to avoid an impact on the system operating safety. Some of these techniques include memory functionality and integrity tests, such as checksums and watchdog times for software processes, including operating system processes. Additionally, software control functions can be performed incrementally rather than in a single step, reducing the potential that a single failure of a software component would cause an unsafe state.

The software safety work activity for level A custom-developed, configurable, and acquired software should fully meet this requirement. For this software type the safety analysis for the software components should be performed. This analysis may be part of the overall safety system analysis if detailed software failures are included. For level A custom-developed safety software, the design concepts that include simplicity of modules that perform safety functions and isolation of those modules should be part of the design considerations. Where the design of the software modules still presents an unacceptable risk to failure of the safety system, fault-tolerant and self-diagnostics designs should be implemented.

Custom-developed, configurable, and acquired level B or level C software applications may be graded. This grading may include fully performing the safety analysis activities for the
software components to ensure the safety aspects are being addressed. The design concepts of simplicity and isolation and fault tolerance and self-diagnostics may not apply to level B or level C software applications, and, thus, can optionally be applied.

This work activity does not apply to utility calculation or commercial design and analysis safety software types.

**Verification and Validation**

Verification and validation is the largest area within the SQA work activities. Verification is performed throughout the life cycle of the safety software. Validation activities are performed at the end of the software development or acquisition processes to ensure the software meets the intended requirements.

Verification and validation activities should be performed by competent staff other than those who developed the item being verified or validated. Verification and validation activities include reviews, inspections, assessments, observations, and testing.

Reviews and inspections of software deliverables requirement specifications, procurement documents, software design, code modules, test results, training materials, user documentation, and processes that guide the software development activities should be performed. The software deliverables may be combined with other software or system documents. Traceability of the software requirements to the software design should be performed. Inspections can be formally implemented Fagan inspections, walkthrough, or desk checks. Verification of the software design should be completed prior to approval of the software for use. This verification may be performed as part of the software development and implementation activity.

Observations and testing can be performed during the development, factory or site acceptance testing, installation, and operation of the software. Software testing activities should be planned and documented. Test cases and procedures, including expected results, should be created. All test activity deliverables should be under configuration management. Test results should be documented and all test activity deliverables placed under configuration management.

Acceptance testing should include functional testing, performance testing, security testing, stress testing, and load testing. Users’ guides, use cases, and operational profiles are instrumental in identifying and detailing the positive test cases and procedures. Failure mode analysis can be used for defining negative test cases and procedures. Testing strategies that may be appropriate for acceptance testing include equivalence class testing, branch and path testing, statistical-based and boundary value testing.

Additionally, the system should continually be monitored to estimate its continuing reliability and safety. Periodic testing of the operational system should be performed to detect any degradation. If testing is not possible, monitoring using quantitative measurement should be performed.
When a new version of a software product is obtained, predetermined and ad hoc test cases and procedures should be performed to validate that the system meets the requirements and does not perform any unintended functions. If the system is operational, only positive testing may be possible. In those instances, it is important to perform analysis of failure modes for the software to understand the consequences if the software or system should get into an abnormal operational state.

Modern utility calculation applications, such as spreadsheet programs, have grown dramatically in power, with a corresponding growth in risk. The addition of macro programming languages and the ability to incorporate add-in programs provide users with nearly the same capabilities as code developed with traditional programming tools. Utility calculation applications are installed on virtually every desktop, and user files containing algorithms and data can be easily modified by users. Calculations performed using applications such as commercial spreadsheet programs may be treated in either of two ways. In the case of relatively straightforward calculations, the calculation result may be checked and verified in the same manner as a hand calculation. For more complex or extensive calculations, where checking and verification of calculation results are impractical or undesirable, the user files containing the calculation formulas, algorithms, or macros should be subject to the entire SLCP. The latter approach may also be expedient for calculation applications that are reused frequently.

Custom-developed software will most likely have a larger number and more detailed deliverables than would utility calculations. For level A safety software all deliverables should be reviewed using V&V methods. Additionally, for level A, traceability of the requirements to the design and from requirements to test cases should be performed. For level B safety software, deliverables that include requirements, test plans and procedures, and test results should be reviewed using V&V methods.

For all level A safety software except utility calculations, acceptance testing work activities should be planned and documented; acceptance test cases and procedures, including expected results should be created; test results should be documented; and all test activity deliverables should be under configuration management. Level A utility calculations and level B and C custom-developed, configurable, acquired, and utility calculations can use a graded approach by applying less formality in the documentation. Simple check lists for acceptance test cases and procedures may be used in place of more detailed test cases and procedures. Test results should be documented and all test activity deliverables placed under configuration management.

For level A software, continual monitoring of safety software operations based on historical failure data and results of periodic reassessment of hazards should be performed. For level A, B, or C software, when new releases of the safety software have been developed, reviews and acceptance testing of changed documents and software should be performed.
**Problem Reporting and Corrective Action**

Coupled with the configuration management of the software system, the problem reporting and corrective action process should address the appropriate requirements of the QAP corrective action system. The reporting and corrective action system will cover 1) methods for documenting, evaluating and corrective software problems; 2) an evaluation process for determining whether a reported problem is indeed a defect or an error; and 3) the roles and responsibilities for disposition of the problem reports, including notification to the originator of the results of the evaluation. If the noted problem is indeed an error, the problem reporting and corrective action system should correlate the error with the appropriate software engineering elements; identify the potential impacts and risks to past, present, and future developmental and operational activities; and support the development of mitigation strategies. After an error has been noted, all users should be apprised to ascertain any impacts on safety basis decisions.

Procurement documents should identify the requirements for suppliers to report problems to the supplier, any required supplier response, and the method for the purchasers to report problems to the supplier.

Maintaining a robust problem reporting and corrective action process is vital to maintaining a reliable and vital safety software system. 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.

The work activity should be fully implemented for all level A and B software types and for level C custom-developed. This formal implementation should include documentation and tracking to closure of any problems reported for the software and authorization to perform the corrective action. A graded approach that reduces the formality of documenting problem reports and approving corrective actions taken may be applied for level A and B utility calculation safety software and all level C software applications except custom-developed. This less formal implementation may include interoffice communications describing the problem identified and the corrective actions planned.

**Training Personnel in the Design, Development, Use, and Evaluation of Safety Software**

Training personnel in designing, developing, testing, evaluating, or using the safety software application is critical for minimizing the consequences of software failure. Although other SQA work activities may indicate that the software satisfies its operational objective, improper or invalid use of the software may negate the safety mitigation strategies included within the software.

Training may be necessary for the analyst, development and test teams, application users, and operations staff. The analyst and developers may need training in fault-tolerant methodologies, safety design methodologies, user interface design issues, testing methodologies, or configuration management to ensure delivery of a robust software application. Meanwhile, the software application users and operations staff may need training specific to the software to ensure that proper data are entered, that proper options and menus
are selected, and that the results of the software can be interpreted correctly. A trained and knowledgeable staff is essential to assess and evaluate the SQA requirements to ensure the proper levels of quality and safety exists in the software.

Training should be commensurate with the scope, complexity, and importance of the tasks and the education, experience, and proficiency of the individual. Personnel should also 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.

Completion of training, education, and/or qualification requirements for all staff involved in the development, testing, use, and evaluation of custom-developed or configurable software graded as level A, B, or C should be documented and reviewed periodically. This may include a position description, qualification criteria, or a list of training courses along with verification of successfully meeting the knowledge requirements. Completion of training, education, and/or qualification requirements for all staff involved in the procurement, testing, use, and evaluation of acquired or utility calculation software graded as level A should be documented and reviewed periodically. For level B and C software applications, this work activity can be graded to include periodic evaluation by the appropriate supervising authority of the training, educational, or qualification requirements for performing assigned tasks associated with using and evaluating acquired or utility calculation software. This work activity does not apply to commercial design and analysis safety software since the training activities associated with commercial design and analysis software are performed by the service supplier.

9. Safety software quality assurance personnel must demonstrate a working level knowledge of software requirements identification and management.

   a. **Explain how software requirements specifications (SRS) are developed and used throughout the software life cycle.**

The following is taken from Institute of Electrical and Electronics Engineers, IEEE-830.

This recommended practice describes recommended approaches for the specification of software requirements.

It is based on a model in which the result of the SRS process is an unambiguous and complete specification document. It should help

- software customers to accurately describe what they wish to obtain
- software suppliers to understand exactly what the customer wants
- individuals to accomplish the following goals:
  - Develop a standard SRS outline for their own organizations
  - Define the format and content of their specific SRSs
  - Develop additional local supporting items such as an SRS quality checklist, or an SRS writer’s handbook
To the customers, suppliers, and other individuals, a good SRS should provide several specific benefits, such as the following:

- Establish the basis for agreement between the customers and the suppliers on what the software product is to do. The complete description of the functions to be performed by the software specified in the SRS will assist the potential users to determine if the software specified meets their needs or how the software must be modified to meet their needs.
- Reduce the development effort. The preparation of the SRS forces the various concerned groups in the customer’s organization to consider rigorously all of the requirements before design begins and reduces later redesign, recoding, and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct.
- Provide a basis for estimating costs and schedules. The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates.
- Provide a baseline for V&V. Organizations can develop their V&V plans much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured.
- Facilitate transfer. The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organization, and suppliers find it easier to transfer it to new customers.
- Serve as a basis for enhancement. Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

b. Define and discuss the SRS attributes as they relate to safety software such as described in DOE G 414.1-4, Safety Software Guide for use with 10 CFR 830 Subpart A, Quality Assurance Requirements and DOE O 414.1C, Quality Assurance and IEEE 830, IEEE Recommended Practice for Software Requirements Specifications.

The following is taken from DOE G 414.1-4.

Maintaining the integrity, safety, and security of all DOE assets and resources is paramount for DOE’s mission. Since software is an integral part of DOE’s resources, the integrity, safety, and security attributes of its software resources are critical to DOE’s mission. All three attributes are interdependent since compromising the security access could obviously present a potential safety hazard. If the integrity of either the data or application itself has been compromised either accidentally or maliciously, safety could be compromised. Therefore when safety software is being addressed, the integrity and security issues should likewise be addressed.

The following is taken from Institute of Electrical and Electronics Engineers, IEEE 830.
**Complete**

An SRS is complete if it includes the following elements:

- All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.
- Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to valid and invalid input values.
- Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.

**Correct**

An SRS is correct if every requirement stated therein is one that the software shall meet. There is no tool or procedure that ensures correctness. The SRS should be compared with any applicable superior specification, such as an SRS, with other project documentation, and with other applicable standards, to ensure that it agrees. Alternatively the customer or user can determine if the SRS correctly reflects the actual needs. Traceability makes this procedure easier and less prone to error.

**Consistent**

Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct.

An SRS is internally consistent if no subset of individual requirements described in it conflicts.

The three types of likely conflicts in an SRS are as follows:

- The specified characteristics of real-world objects may conflict. For example,
  - The format of an output report may be described in one requirement as tabular but in another as textual.
  - One requirement may state that all lights shall be green while another may state that all lights shall be blue.
- There may be logical or temporal conflict between two specified actions. For example,
  - One requirement may specify that the program will add two inputs and another may specify that the program will multiply them.
  - One requirement may state that A must always follow B, while another may require that A and B occur simultaneously.
- Two or more requirements may describe the same real-world object but use different terms for that object. For example, a program’s request for a user input may be called a “prompt” in one requirement and a “cue” in another. The use of standard terminology and definitions promotes consistency.
Clear and Unambiguous

An SRS is clear and unambiguous if every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term.

In cases where a term used in a particular context could have multiple meanings, the term should be included in a glossary where its meaning is made more specific.

An SRS is an important part of the requirements process of the SLC and is used in design, implementation, project monitoring, V&V and in training. The SRS should be unambiguous both to those who create it and to those who use it. However, these groups often do not have the same background and therefore do not tend to describe software requirements the same way. Representations that improve the requirements specification for the developer may be counterproductive in that they diminish understanding to the user and vice versa.

Verifiable

An SRS is verifiable if every requirement stated therein is verifiable. A requirement is verifiable if there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable.

Non-verifiable requirements include statements such as “works well,” “good human interface,” and “shall usually happen.” These requirements cannot be verified because it is impossible to define the terms “good,” “well,” or “usually.” The statement that “the program shall never enter an infinite loop” is non-verifiable because the testing of this quality is theoretically impossible.

Feasible

The following is taken from Search Software Quality.com, Defining Requirements during Software Project Feasibility Analysis.

Feasibility analysis answers two main questions:
1. Can this approach to the solution work?
2. Is the approach worthwhile, that is, do its benefits exceed its costs?

Requirements are the basis for answering both these questions. A solution works because it satisfies the requirements. Satisfying the requirements provides measurable value by solving a problem, taking an opportunity, or meeting a challenge. A solution provides value if it satisfies the requirements.

Feasibility analysis evaluates the ability of alternative approaches, not products, to economically meet requirements, which are the real business requirements—business deliverable “whats” that provide value when delivered by the approach’s product “how.”
Many requirements definitions are only high-level, especially those characterized as “business requirements,” which often are nothing more than a few sentences describing objectives or expected benefits. Such seemingly high-level requirements are actually a trap for several reasons. Most often they are high-level product requirements rather than high-level business requirements.

Objectives or expected benefits are not the requirements. Instead, they are what will be gained if the requirements are met. Moreover, objectives by themselves tell us nothing about what’s causing our current results. Consequently, objectives do not provide a suitable basis for identifying solutions which reasonably will achieve the expected benefits by meeting the objectives.

Regardless of whether the feasibility analysis is performed consciously or unconsciously, formally or informally, the common failure to adequately define high-level real business requirements frequently initiates projects that are already doomed by the time a project manager or business analyst is assigned. Without a reliable definition of what must be delivered to provide value, not only will the project be unable to deliver expected value, but there’s no meaningful basis for estimating what it will take to deliver a product that achieves the value.

c. Describe the purpose, scope, and content of SRS and requirements traceability matrix and describe the methods to ensure that all elements of the SRS are addressed.

The following is taken from DOE G 200.1-1, chapter 4.

An RTM is a table used to trace project life cycle activities and work products to the project requirements.

Every project requirement must be traceable back to a specific project objective(s) described in the project plan. This traceability ensures that the product will meet all of the project objectives and will not include inappropriate or extraneous functionality.

All work products developed during the design, code, and testing processes in subsequent life cycle stages must be traced back to the project requirements described in the SRS. This traceability ensures that the product will satisfy all of the requirements and remain within the project scope.

It is also important to know the source of each requirement, so that the requirements can be verified as necessary, accurate, and complete. Meeting conference records, user survey responses, and business documents are typical sources for project requirements.

Work Product: Develop a matrix to trace the requirements back to the project objectives identified in the project plan and forward through the remainder of the project life cycle stages. Place a copy of the matrix in the project file. Expand the matrix in each stage to show traceability of work products to the requirements and vice versa.
One example of a RTM is available in table 1. A second method for tracing requirements is a threading matrix that groups requirements by project objectives. Under each project objective, the source of the requirement, the unique requirement identification number, and the life cycle activities are listed in columns along the top and the project requirements in rows along the left side. As the project progresses through the life cycle stages, a reference to each requirement is entered in the cell corresponding to the appropriate life cycle activity. An example of this type of RTM is illustrated in table 3.

**Table 3. Sample RTM**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The software product shall have four user access levels with the capability to add new access levels in the future.</td>
<td>conference record dated 5/19/95</td>
<td>SYSADM 1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Each user access level shall have a unique designation.</td>
<td>conference record dated 5/19/95</td>
<td>SYSADM 1.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One user access level shall allow read-only access to the production database.</td>
<td>conference record dated 5/19/95</td>
<td>SYSADM 1.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The second user access level shall allow read and write access to the production database.</td>
<td>conference record dated 5/19/95</td>
<td>SYSADM 1.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The third user access level shall allow read, write, and delete access to the production database and read-only access to the history database.</td>
<td>conference record dated 5/19/95</td>
<td>SYSADM 1.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The fourth user access level shall allow read, write, and delete access to all application databases.</td>
<td>conference record dated 5/19/95</td>
<td>SYSADM 1.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: DOE G 200.1-1, chapter 4

d. Describe and give specific examples of the following software requirements such as described in DOE G 414.1-4:

- Functional
- Performance
- Access control
- Interface with safety requirements
- Installation considerations
- Design constraints

The following descriptions are taken from DOE G 200.1-1, chapter 4.

**Functional**

Functional requirements define what the software product must do to support the system owner’s business functions and objectives. The functional requirements should answer the following questions:

- How are inputs transformed into outputs?
- Who initiates and receives specific information?
- What information must be available for each function to be performed?

Identify requirements for all functions whether they are to be automated or manual. Describe the automated and manual inputs, processing, outputs, and conditions for all functions.
Include a description of the standard data tables and data or records that will be shared with other applications. Identify the forms, reports, source documents, and inputs/outputs that the software product will process or produce to help define the functional requirements.

A functional model should be developed to depict each process that needs to be included. The goal of the functional model is to represent a complete top-down picture of the software product.

Flow diagrams should be used to provide a hierarchical and sequential view of the system owner’s business functions and the flow of information through the processes.

**Performance**

Performance requirements define how the software product must function (e.g., hours of operation, response times, and throughput under detailed load conditions). The information gathered in defining the project objectives can translate into very specific performance requirements. Also, government and DOE policy can dictate specific availability and response times.

**Access Control**

The following list provides sample questions that can be used to help define the access controls for the software product:

- What access restrictions are placed on the users by their organization or programmatic office?
- What are the audit and other checking needs for the software product?
- What separation of duties, supervisory functions related to control, operating environment requirements, or other functions will impact the software product?
- What measures will be used to monitor and maintain the integrity of the software product and the data from the user’s viewpoint?

**Interface with Safety Requirements**

The hardware and software interface requirements must specify hardware and software interfaces required to support the development, operation, and maintenance of the software product. The following information should be considered when defining the hardware and software interface requirements:

- System owner’s and users’ computing environment
- Existing or planned software that will provide data to or accept data from the software product
- Other organizations or users having access to the software product
- Purpose or mission of interfacing software
- Common users, data elements, reports, and sources for forms/events/outputs
- Timing considerations that will influence sharing of data, direction of data exchange, and security constraints
- Development constraints such as the operating system, database management system, language compiler, tools, utilities, and network protocol drivers
Standardized system architecture defined by hardware and software configurations for organizations, programmatic offices, or telecommunications programs

Installation Considerations
Identify the installation requirements for any new hardware, operating system, or software. For hardware installations, consider environmental factors such as air-conditioning, power supply, and security requirements. For software installations, consider proprietary software such as database management systems. For application software, consider the installation of the application’s programs, parallel operation of the old and new applications, or the cutover from a test to a production environment. Hardware and software installation must be coordinated with the work cycles of the user organization to create a minimum of disruption, and to ensure that data are available as needed. Installation must be scheduled to ensure that, when data conversion is necessary, the needed data are protected.

Design Constraints
Document all design constraints including processing, performance, interface, resource, safety, security and reliability requirements. Define data constraints such as limits, formats, messages, commands, and displays.

10. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the safety software design concepts.

a. Discuss the following software design concepts as they relate to safety software such as described in part in IEEE 1016, IEEE Recommended Practice for Software Design Descriptions:
   - Modular design
   - External interface
   - Interfaces between both safety and non-safety components
   - Interface integrity
   - Data integrity
   - Flow control
   - Exception and error handling
   - Simplicity
   - Decoupling
   - Isolation
   - Software failure mode analyses

Modular Design
The following is taken from University of Florida, Computer & Information Science & Engineering, PASCAL Programming: Section 3, Modular Software Design.

In order to produce programs that are readable, reliable, and can be easily maintained or modified, one must use modular software design. This means that, instead of having a large collection of statements strung together in one partition of in-line code, we segment or divide
the statements into logical groups called modules. Each module performs one or two tasks, and then passes control to another module. By breaking up the code into bite-sized chunks we are able to better control the flow of data and control. This is especially true in large software systems or systems that have safety implications.

External Interface

The following is taken from Open Process Framework Repository Organization, External Interface Specifications.

External interface specifications are the requirements work product produced during system or application development that formally specifies the interfaces to all external systems and applications.

The typical objectives of an external interface specification are to perform the following:

- Help establish a consensus among project stakeholders concerning the required system:
  - Data interfaces and protocols
  - Hardware interfaces and protocols
  - Software interfaces and protocols
- Baseline these requirements for validation and acceptance.
- Be a basis for cost and schedule estimates.
- Be a basis for future extensions and enhancements.

The typical benefits of an external interface specification are the following:

- Decrease the time required to integrate the application with existing applications and hardware.
- Increase the quality of the integration between the application and existing non-human externals.

An external interface specification typically can be started if the following preconditions hold:

- The requirements team is staffed and adequately trained in requirements specification.
- Domain experts who know the interfaces are available.
- The SRS has been started.

The following guidelines apply.

Do not produce an external interface specification unless the application must interface with an external systems or applications.

Where practical, specify open interface standards with the following decreasing order of preference:

1. International interface standards
2. De facto interface standards
3. Proprietary corporate standards
4. Proprietary program standards

Consider including the specification of external interfaces in an additional section of the SRS rather than in a separate document if there are few mandatory external interfaces and these interfaces are simple.

Where practical, specify external interfaces by reference to the published standards or existing documentation about individual external applications rather than redundantly repeating the information.

Do not use this specification to specify the internal interfaces of components of the application unless use of these components has been required as a design constraint in the SRS.

*Interfaces between both Safety and Non-Safety Components*

Safety Components
The following is taken from Joint Services Computer Resources Management Group, *Software System Safety Handbook*.

System safety engineering is a proven and credible function supporting the design and systems engineering process. The steps in the process for managing, planning, analyzing, and coordinating system safety requirements are well established. The general system safety program requirements are as follows:

- Eliminate identified hazards or reduce associated risk through design, including material selection or substitution.
- Isolate hazardous substances, components, and operations from other activities, areas, personnel, and incompatible materials.
- Locate equipment so that access during operations, servicing, maintenance, repair, or adjustment minimizes personnel exposure to hazards.
- Minimize risk resulting from excessive environmental conditions.
- Design to minimize risk created by human error in the operation and support of the system.
- Consider alternate approaches to minimize risk from hazards that cannot be eliminated. Such approaches include interlocks; redundancy; fail-safe design; fire suppression; and protective clothing, equipment, devices, and procedures.
- Protect power sources, controls, and critical components of redundant subsystems by separation or shielding.
- Ensure personnel and equipment protection provide warning and caution notes in assembly, operations, maintenance, and repair instructions as well as distinctive markings on hazardous components and materials, equipment, and facilities. These shall be standardized in accordance with requirements.
- Minimize severity of personnel injury or damage to equipment in the event of a mishap.
- Design software-controlled or monitored functions to minimize initiation of hazardous events or mishaps.

*Interface Integrity*

The following is taken from DOE G 200.1-1, chapter 4.

The user interface requirements should describe how the user will access and interact with the software product, and how information will flow between the user and the software product.

**Interface Issues:** The following are some of the issues that should be considered when trying to identify user interface requirements:

- The users’ requirements for screen elements, navigation, and help information
- The standards for the programmatic organization, DOE, government, and industry that apply to user interfaces
- The range of skill levels of the users who will access and use the software product
- The range of work that the users will be performing with the software product

Define the user interface requirements by identifying and understanding what is most important to the user, not what is most convenient for the project team. Work with the system owner and users to develop a set of user interface requirements that can be used for all automated products for the system owner’s organization. A standard set of user interface requirements will simplify the design and code processes, and ensure that all automated products have a similar look and feel to the users. When other constraints do not permit the use of existing user interface standards, an attempt should be made to keep the user interface requirements as close as possible to the existing standard.

*Data Integrity*

The following is taken from DOE G 200.1-1, chapter 4.

If a software product under development processes sensitive personal information, appropriate safeguards must be established to protect the information from accidental disclosure.

Implement applicable security procedures to ensure data integrity and protection from unauthorized disclosure, particularly during development efforts. The organization that owns the data defines the data classification. The project team must be aware of all the types of data and of any classified or proprietary algorithms used in the software product.

**Procedure:** Use the following procedure to determine computer security requirements:

- Identify the types of data that will be processed by the software product.
- Determine preliminary data protection requirements.
- Coordinate with the owner of the host platform to identify existing supporting computer security controls, if applicable.
- Incorporate security requirements into the SRS.
Maintain a record of all security and access requirements. Save for incorporation into the SRS. Place a copy of the security and access requirements in the project file.

The following list provides sample questions that can be used to help define the access controls for the software product:

- What access restrictions are placed on the users by their organization or programmatic office?
- What are the audit and other checking needs for the software product?
- What separation of duties, supervisory functions related to control, operating environment requirements, or other functions will impact the software product?
- What measures will be used to monitor and maintain the integrity of the software product and the data from the user’s viewpoint?

**Flow Control**

The following is taken from DOE G 200.1-1, chapter 5.

The logical model defines the flow of data through the software system and determines a logically consistent structure for the software. Each module that defines a function is identified, interfaces between modules are established, and design constraints and limitations are described. The focus of the logical model is on the real-world problem or need to be solved by the software product.

A logical model has the following characteristics:

- Describes the final sources and destinations of data and control flows crossing the system boundary rather than intermediate handlers of the flows
- Describes the net transfer of data across the system boundary rather than the details of the data transfer
- Provides for data stores only when required by an externally imposed time delay

When building a logical model, the organization of the model should follow the natural organization of the software product’s subject matter. The names given to the components of the model should be specific. The connections among the components of the model should be as simple as possible.

The logical model should be documented in user terminology and contain sufficient detail to obtain the system owner’s and users’ understanding and approval. Use DFDs to show the levels of detail necessary to reach a clear, complete picture of the software product processes, data flow, and data stores.

Maintain the logical model and DFDs for incorporation into the functional design document. Place a copy of the logical model and DFDs in the project file. Keep the logical model and diagrams up-to-date. They will serve as a resource for planning enhancements during the maintenance stage, particularly for enhancements involving new functions.
**Exception and Error Handling**

The following is taken from DOE G 200.1-1, chapter 5.

System messages are the various types of information that the system provides to the user such as status messages, user prompts, and error messages.

Status messages are important for giving users the feeling they are in control of the software. They tell users what the software is doing, where they are in the sequence of screens, what options they have selected, and what options are available.

Prompts inform the user to type data or commands or to make a simple choice. Use prompts to ask the user to make a simple choice or to enter data or commands. Be as specific as possible. Include memory aids in the prompt to help users type a response in the proper format and order, initiate infrequently used processes, or clearly identify exceptions to normal practice. When defaults are allowed with prompts, indicate clearly which default value will be initiated.

Error messages should allow users to recover from mistakes by making it clear what the mistake was and how to correct it. Error messages need to be specific about why a mistake was made.

- Design the software product to check for obvious errors.
- Be as specific as possible in describing the cause of an error. Do not use error codes.
- Do not assign blame to the user or the software in an error message. Use a neutral tone.
- Whenever possible, the error message should indicate what corrective action the user needs to take.
- Be consistent in the format, wording, and placement of messages.
- Consider describing error messages at more than one level of detail.

**Simplicity, Decoupling, and Isolation**

The following is taken from DOE G 414.1-4.

The design of the software is critical to ensuring safe operation of the system. The software design should consider principles of simplicity, decoupling, and isolation to eliminate the hazards. Complexity of the software design, including the logic and number of data inputs, has proven to increase the defect density in software components. The safety features should be separate from non-safety modules, minimizing the impact of failure of one module on another. The interfaces between the modules need to be defined and tested thoroughly. Separation of the safety features also allows for more rigorous software development and verification practices to be applied to the safety components while providing the appropriate and cost-effective level of SQA applied to the non-safety components. Software engineering safety design practices should include process flow analysis, data flow analysis, path analysis, interface analysis, and interrupt analysis during the design phase.
Software Failure Mode Analyses
The following is taken from DOE G 414.1-4.

During the initial concept and requirement analysis phases for the software, potential failures need to be identified and evaluated for their consequences of failure and probability of occurrence. Some potential problems 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.

There are several hazard analysis techniques that may be used for this purpose. Many of these techniques are performed as preliminary analyses and later updated as more information is known about the requirements and design structure. These techniques include failure modes and effects analysis, fault-tree modeling, event-tree modeling, cause-consequence diagrams, hazard and operability analysis, and interface analysis. Techniques such as these should be applied and appropriately documented to understand and assess the impact of software failures on the system.

11. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of the safety software design and implementation practices.

   a. Discuss the following concepts as they relate to safety software coding:
      ▪ Development environment
      ▪ Target environments and reusable components
      ▪ Data structure
      ▪ Logic structure
      ▪ Embedded comments
      ▪ Peer reviews

Development Environment
The following is taken from DOE G 200.1-1, chapter 7.

Developing the programming environment involves assembling and installing the hardware, software, telecommunications equipment, databases, and other items required to support the programming effort. When the installation of the equipment or software is complete, conduct testing to verify the operating characteristics and functionality of the hardware and software. If required, security software and procedures should be activated when the installations are completed.

If the operational environment is also the programming environment, it may be necessary to alter the operational environment to accommodate an infrastructure of purchased hardware and software for use during programming and testing. Before being integrated into, or used to support, the software product, vendor products should be tested to verify that the product satisfies the following objectives:
   ▪ The product performs as advertised/specified.
   ▪ The product’s performance is acceptable and predictable in the target environment.
- The product fully or partially satisfies the project requirements.
- The product is compatible with the project team’s other hardware and software tools.

Time should be planned for the project team to become familiar with new products. Ensure that the project team members who will use the hardware or software obtain proper training. This may involve attendance at formal training sessions conducted by the vendor or the services of a consultant to provide in-house training.

This is a good time to review the programming standards that were established in the system design stage. Make any changes to the standards that are needed to accommodate the procured hardware and software.

**Target Environment and Reusable Components**
The following is taken from DOE G 200.1-1, chapter 3.

If the requirements are met by reusable software obtained from: the microcomputer applications system library, the systems review inventory system, or the survey of administrative systems at DOE reporting system, then adapting that software may provide the best solution. If the source application is very customized to meet specific requirements, was developed to standards, and includes documentation, this can be a very attractive alternative to a custom-built application. Reusable code may be customized to meet specific requirements.

Reusable code can be either modules of code that are used as particular software languages administrators have written, or units of code that are reengineered to perform a similar function. If the reusable tools for the particular development language are numerous, identifying software libraries available for particular software languages may be more attractive than modifying an application that will need considerable customization.

Some software engineering teams develop modules of reusable code that are language specific and perform functions such as screen formats, data validation, error-handling, data access, and other frequently used routines. Once developed, tested, and refined, these routines become reliable building blocks for the rapid development of other applications.

**Data Structure**
The following is taken from DOE G 200.1-1, chapter 6.

The goal of this stage is to translate the user-oriented functional design specifications into a set of technical, computer-oriented system design specifications; and to design the data structure and processes to the level of detail necessary to plan and execute the programming and installation stages.

General module specifications should be produced to define what each module is to do, but not how the module is to be coded. Effort focuses on specifying individual routines and data structures while holding constant the software structure and interfaces developed in the previous stage. Each module and data structure is considered individually during detailed
design with emphasis placed on the description of internal and procedural details. The primary work product of this stage is a software system design that provides a blueprint for the coding of individual modules and programs.

**Logic Structure**
The following is taken from eHow, How to Know the Fundamentals of Program Structure.

The logical structure is the organization of the source code. It affects runtime performance. In a program that is designed according to the object-oriented model, the physical and logical structures correlate closely since a class (logical unit) corresponds to a module (physical unit).

The following guidance steps are related to building a logic structure:

1. Enforce strict rules in the code by using variable identifiers and modifiers as appropriate. If a variable is read-only, use const. To control the lifetime of a variable, use static. If an integer is iterated from 0 to 99, make it unsigned.
2. Break the program into small chunks of code and keep the functions small. Keep each function to less than five lines (between the braces) if possible. Give the function a meaningful label.
3. Group the functions into name spaces to reduce ambiguity and prevent name duplication.
4. Use inheritance when doing object-oriented programming to mirror the structure of real-life hierarchies. Abstract classes should represent general concepts like obstacle and concrete classes should represent specific objects.
5. Employ polymorphism to both data types and functions so as to create generic programming. When functions and data types are written generically to eliminate dependence on type, the coding becomes more flexible. Make sure to fully maintain static-type safety.
6. Use the most appropriate design pattern for the project if it is big enough. Design patterns are proven tested template solutions that address the most commonly encountered software engineering problems. Design patterns emphasize robustness and scalability. Scalability means that the design makes adding new components as the user requirements increase easy without modifying the original source code.

**Embedded Comments**
The following is taken from Noveedge, Maya Help, Embedded Comments.

An embedded comment is text in a Maya embedded language (MEL) file that is ignored when the file is read in.

If a line in the file contains two consecutive slashes (//"), everything from there to the end of the line is considered to be a comment. This is commonly known as a C++ style comment.
If a line in the file contains a slash followed by an asterisk (/*), everything from there on is considered to be comment, until the next occurrence of an asterisk followed by a slash (*). This is commonly known as a C style comment.

**Note**: Although supported in MEL, C style comments are not generally used.

**Peer Reviews**

The following is taken from Seven Truths About Peer Reviews by Karl E. Wiegers.

Peer reviews have long been recognized as a powerful way to improve quality, yet few software organizations have systematic and effective review programs in place. A quality-driven organization will practice a variety of peer review methods, spanning a spectrum of formality, rigor, effectiveness, and cost. The following are descriptions of some common review approaches.

An inspection is the most systematic and rigorous type of peer review. Inspection follows a well-defined multistage process with specific roles assigned to individual participants. Inspections are more effective at finding defects than are informal reviews.

Team reviews are planned and structured but less formal and less rigorous than inspections. Typically, the overview and follow-up inspection stages are simplified or omitted, and some participant roles may be combined.

A walkthrough is an informal review in which the work product’s author describes it to selected colleagues and solicits comments. Walkthroughs differ significantly from inspections because the author takes the dominant role; other specific review roles are usually not defined. Walkthroughs are informal because they typically do not follow a defined procedure, do not specify exit criteria, require no management reporting, and generate no metrics.

In pair programming, two developers work on the same program simultaneously at a single workstation, continuously reviewing their joint work. Pair programming lacks the outside perspective of someone who is not personally attached to the code that a formal review brings.

In a peer desk check, only one person besides the author examines the work product. A peer desk check typically is an informal review, although the reviewer could employ defect checklists and specific analysis methods to increase effectiveness.

A pass around is a multiple, concurrent peer desk check, in which several people are invited to provide comments. The pass around mitigates two major risks of a peer desk check: the reviewer failing to provide timely feedback and the reviewer doing a poor job.

The project team should select the cheapest review method that will reduce the risk associated with defects remaining in a given deliverable to an acceptable level. Use inspections for high risk work products, and rely on cheaper techniques for components that
have lower risk. Although there are many ways to get colleagues to help improve the deliverables, only inspections are a software industry best practice.

b. Discuss the following documents and describe how each supports safety software coding:
   - Design specifications
   - Program specifications
   - Coding standards
   - System design document
   - Programmers manual
   - Users manual

Design Specifications
The following is taken from DOE G 200.1-1, chapter 6.

During the functional design stage, a decomposition of the software product requirements results in a collection of design entities. In the system design stage, these design entities are grouped into the routines, modules, and programs that need to be developed or acquired as COTS or reusable software.

Expand the functional design to account for each major software action that must be performed and each data object to be managed. Detail the design to a level such that each program represents a function that a programmer will be able to code.

Use the following procedure to design the software module specifications:
1. Identify a software program for each action needed to meet each function or data requirement in the SRS and the data dictionary.
2. Identify any routines and programs that may be available as reusable code or objects from existing applications or COTS software. The system review inventory system maintained at DOE Headquarters and the Energy Science and Technology Software Center (ESTSC) located at Oak Ridge, Tennessee are recommended sources for identifying reusable software. The ESTSC is the Department’s central collection of DOE-supported software packages. The center also collects software from the Nuclear Regulatory Commission and others, and maintains contact with other software centers.
3. Identify programs that must be designed and developed.
4. Assign a name to each program and object that is functionally meaningful.
5. Identify the system features that will be supported by each program.
6. Specify each program interface. Update the data dictionary to reflect all program and object interfaces changed while evolving from the functional to the system design.
7. Define and design significant attributes of the programs to be custom built.
8. Expand the program interfaces to include control items needed for design validity.
9. Combine similar programs and objects. Group the design entities into modules based on closely knit functional relationships. Formulate identification labels for these modules.
10. Show dependencies between programs and physical data structures. Avoid defining a program that not only needs data residing in a file or global table, but also depends on the physical structure or location of data.

11. Change the design to eliminate features that reduce maintainability and reusability.

Document the system design primarily in the form of diagrams. Supplement each diagram with text that summarizes the function and highlights important performance and design issues.

When using structured design methods, the design diagrams should
- depict the software as a top-down set of diagrams showing the control hierarchy of all software programs to be implemented
- define the function of each software program
- identify data and control interfaces between programs
- specify files, records, and global data accessed by each program

When using object-oriented or data-centered design methods, the design diagrams should
- show the data objects to be managed by the software
- specify the program functions to be included within each object
- identify functional interfaces between objects
- specify files and records comprising each object
- identify relationships between data files

Program Specifications
The following is taken from DOE G 200.1-1, chapter 6.

A program specification is a written procedural description of each software system routine. The program specification should provide precise information needed by the programmers to develop the code.

Many techniques are available for specifying the system design, such as formal specification languages, program design languages, meta-code, tabular tools, and graphical methods. In object-oriented design, the specification of requirements and preliminary design constraints and dependencies often results in the design language producing the detailed specifications.

Select the technique or combination of techniques that is best suited to the software project and to the experience and needs of the programmers who will use the system design as their blueprint. The following are suggestions for using the techniques:
- Decision trees are useful for logic verification or moderately complex decisions that result in up to 10-15 actions. Decision trees are also useful for presenting the logic of a decision table to users.
- Decision tables are best used for problems involving complex combinations of up to 5-6 conditions. Decision tables can handle any number of actions; however, large numbers of combinations of conditions can make decision tables unwieldy.
- Structured English is best used wherever the problem involves combining sequences of actions with decisions or loops. Once the main work of physical design has been
done and physical files have been defined, it becomes extremely convenient to be able to specify physical program logic using the conventions of structured English, but without getting into the detailed syntax of any particular programming language.

- Standard English is best used for presenting moderately complex logic once the analyst is sure that no ambiguities can arise.

**Coding Standards**

The following is taken from DOE G 200.1-1, chapter 7.

The following coding practices should be implemented:

- The programming staff should meet at scheduled intervals to discuss problems encountered and to facilitate program integration and uniformity.
- Program uniformity should be achieved by using a standardized set of naming conventions for programs, data elements, variables, and files.
- Modules that can be shared by programs requiring the same functionality should be implemented to facilitate development and maintenance.
- Meaningful internal documentation should be included in each program.
- All code should be backed up on a daily basis and stored in an offsite location to avoid catastrophic loss.
- A standard format for storing and reporting elements representing numeric data, dates, times, and information shared by programs should be determined.
- The system design document should be updated to reflect any required deviations from the documented design.

**System Design Document**

The following is taken from DOE G 200.1-1, chapter 6.

The system design document records the results of the system design process and describes how the software product will be structured to satisfy the requirements identified in the SRS. The system design document is a translation of the requirements into a description of the software structure, software components, interfaces, and data necessary to support the programming process.

Prepare the system design document and submit it to the system owner and users for their review and approval. The approved system design document is the official agreement and authorization to use the design to build the software product. Approval implies that the design is understood, complete, accurate, and ready to be used as the basis for the subsequent life cycle stages. Place a copy of the approved system design document in the project file.

**Programmer’s Manual**

The following is taken from DOE G 200.1-1, chapter 7.

The programmer’s reference manual contains programming information used by the maintenance staff to maintain the programs, databases, interfaces, and operating environment. The programmer’s reference manual should provide an overall conceptual
understanding of how the software product is constructed and the details necessary to implement corrections, changes, or enhancements.

The programmer’s reference manual describes the logic used in developing the software product and the functional and system flow to help the maintenance programmers understand how the programs fit together. The information should enable a programmer to determine which programs may need to be modified to change a system function or to fix an error.

The following are typical features of a programmer’s reference manual:

- A description of the technical environment, including versions of the programming language(s) and other proprietary software packages
- A brief description of the design features, including descriptions of unusual conditions and constraints
- An overview of the software architecture, program structure, and program calling hierarchy
- The design and programming standards and techniques used to develop the software product
- Concise descriptions of the purpose and approach used for each program
- Layouts for all data structures and files used in the software product
- Descriptions of maintenance procedures, including configuration management, program checkout, and system build routines
- The instructions necessary to compile, link, edit, and execute all programs

User’s Manual

The following is taken from DOE G 200.1-1, chapter 7.

The user’s manual provides detailed information users need to access, navigate, and operate the software product. Users rely on the user’s manual to learn about the software or to refresh their memory about specific functions. A user’s manual that is organized functionally so that the information is presented the same way the software product works helps users understand the flow of menus and options to reach the desired functions.

Different categories of users may require different types of information. A modular approach to developing the user’s manual to accommodate the needs of different types of users eliminates duplication and minimizes the potential for error or omission during an amendment or update. For example, separate general information that applies to all users from the special information that applies to selected users such as system administrators or database administrators. The special information can be presented in appendices or supplements that are only provided to the users who need the information.

Write the draft user’s manual in clear, nontechnical terminology that is oriented to the experience levels and needs of the user(s). The following are typical features of a user’s manual:
Overview information on the history and background of the project and the architecture, operating environment, and current version or release of the software product
- Instructions for how to install, setup, or access the software product
- Complete coverage of all software functions, presented in a logical, hierarchical order
- Accurate pictures of screens and reports, ideally with data values shown, so the user can easily relate to examples
- In-depth examples and explanations of the areas of the software product that are most difficult to understand
- Clear delineation of which features are accessible only to specific users
- Instructions on accessing and using online help features
- Procedures for data entry
- Descriptions of error conditions, explanations of error messages, and instructions for correcting problems and returning to the function being performed when the error occurred
- Instructions for performing queries and generating reports
- Who to contact for help or further information

12. Safety software quality assurance personnel must demonstrate a working level knowledge of the safety software verification and validation processes that ensure software will adequately fulfill all intended safety functions.

a. Describe the following processes and documents as they relate to safety software verification and validation such as described in IEEE 1012, *IEEE Standard for Software Verification and Validation*:

- Validation of requirements
- Verification and validation of design
- Verification and validation of source code
- Unit/component testing
- Integration testing
- System testing
- Verification and validation plan test cases
- Verification and validation reports
- Verification and validation of tools
- Independent verification and validation
- Acceptance testing
- Installation and checkout testing

*Validation of Requirements*

The following is taken from IEEE 1012.

The requirements V&V activity addresses software requirements analysis of the functional and performance requirements, interface external to the software, and requirements for qualification, safety and security, human factors engineering, data definitions, user...
documentation for the software, installation and acceptance, user operation and execution, and user maintenance. V&V test planning begins during the requirements V&V activity and spans several V&V activities.

The objectives of the requirements V&V is to ensure the correctness, completeness, accuracy, testability, and consistency of the system software requirements.

Verification and Validation of Design
The following is taken from the Institute of Electrical and Electronics Engineers, IEEE 1012.

In software design, software requirements are transformed into an architecture and a detailed design for each software component. The design includes databases and system interfaces. The design V&V activity addresses software architectural design and software detailed design. V&V test planning continues during the design V&V activity.

The objective of design V&V is to demonstrate that the design is a correct, accurate, and complete transformation of the software requirements and that no unintended features are introduced.

Verification and Validation of Source Code
The following is taken from the Institute of Electrical and Electronics Engineers, IEEE 1012.

In software implementation, the system design is transformed into code, database structures, and related machine executable representations. The implementation V&V activity addresses software coding and testing, including the incorporation of reused software products. The objective of implementation V&V is to verify and validate that these transformations are correct, accurate, and complete.

Unit/Component Testing
The following is taken from DOE G 200.1-1, chapter 4.

The unit/component test phase involves testing of the individual software units or groups of related units. A unit is a component that is not subdivided into other components; it is a logically separable part of a computer program. Evaluate each unit of code on how well it meets the performance requirements for which it was designed.

Consider timing, memory, accuracy in producing numerical and logical results; and the preparation of input and output required for validating program logic, syntax, and performance requirements. This test phase is performed by the programmer(s) responsible for writing the code.

Integration Testing
The following is taken from DOE G 200.1-1, chapter 4.
Integration testing is an orderly progression of testing in which software elements, hardware elements, or both are combined and tested to evaluate the interaction between them. Each program/module must be tested. Integration testing is required to validate that groups of related programs, when combined to establish an integrated functional module of code, interface properly, and perform the software functions for which they were designed. Examine the source program/module statements to ensure that the program logic meets the requirements of the design and that the application satisfies an explicit functional requirement. This test phase is performed by the project team.

System Testing
The following is taken from DOE G 200.1-1, chapter 4.

The system test phase tests the integrated hardware and software to verify that the software product meets its specified requirements and operates successfully on the host platform. This test phase is required to validate, when the entire software product is loaded onto the host platform, that the proper initialization is performed; decision-branching paths are appropriate; and all software functions are performed as specified in the SRS. System testing validates that the software product produces the required outputs and interfaces properly with other systems with which the software product gives or receives data; that transaction response times meet user expectations; and machine resource allocation and utilization are within expected norms. This test phase can be performed by the project team or by an independent test group with support from the project team.

Verification and Validation Test Cases
The following is taken from DOE G 414.1-4.

Verification and validation is the largest area within the SQA work activities. Verification is performed throughout the life cycle of the safety software. Validation activities are performed at the end of the software development or acquisition processes to ensure the software meets the intended requirements.

Verification and validation activities should be performed by competent staff other than those who developed the item being verified or validated. Verification and validation activities include reviews, inspections, assessments, observations, and testing.

Test cases and procedures, including expected results, should be created. All test activity deliverables should be under configuration management. Test results should be documented and all test activity deliverables placed under configuration management.

Acceptance testing should include functional testing, performance testing, security testing, stress testing, and load testing. Users’ guides, use cases, and operational profiles are instrumental in identifying and detailing the positive test cases and procedures. Failure-mode analyses can be used for defining negative test cases and procedures.
When a new version of a software product is obtained, predetermined and ad hoc test cases and procedures should be performed to validate that the system meets the requirements and does not perform any unintended functions.

Acceptable methods for evaluating the software test case results include 1) analysis without computer assistance, 2) other validated computer programs, 3) experiments and test, 4) standard problems with known solutions, and 5) confirmed published data and correlations.

**Verification and Validation Reports**

The following is taken from IEEE 1012.

Verification and validation reporting occurs throughout the SLC. The V&V effort shall produce the required outputs for each V&V task performed. The format and grouping of the reports may be user defined. The V&V reports should include the following:

- Anomaly evaluation
- Concept documentation evaluation
- Configuration management assessment
- Contract verification
- Criticality analysis
- Evaluation of new constraints
- Hardware/software/user requirements allocation analysis
- Hazard analysis
- Installation checkout
- Installation configuration audit
- Interface analysis
- Migration assessment
- Operating procedures evaluation
- Proposed change assessment
- Recommendations
- Retirement assessment
- Review results
- Risk analysis
- Security analysis
- Software design evaluation
- Software requirements evaluation
- Source code and source code documentation evaluation
- System requirements review
- Test results
- Traceability analysis

An activity summary report should summarize the results of V&V tasks performed for the following V&V life cycle activities:

- Acquisition support
- Planning
- Concept
- Requirements
- Design
- Implementation
- Test
- Installation and checkout
- Operation
- Maintenance

For the operation and maintenance life cycle activities, V&V activity summary reports may be either updates to previous V&V activity summary reports of separate documents.

**Verification and Validation of Tools**

The following is taken from the International Atomic Energy Agency, Safety Standard Series No. NS-G.1-1.

The tools to be used should be identified in the development plan. The tools should be selected so as to facilitate the proper application of the methods, standards and procedures selected. Planning in advance for the entire project will help in the selection of an integrated set of tools. The tools should be properly qualified for their function in the development, management or verification of the system. The correctness of their output should be ensured by a tool certification process, by means of cross-verification or by reverse engineering.

The tools employed in producing software belong to the two following broad categories: 1) Software development tools, whose output becomes part of the program implementation and which can therefore introduce errors. Code generators, compilers and linkers are examples. 2) Software verification tools that cannot introduce errors. Tools for static analysis and test coverage monitors are examples.

For a tool of either category to be employed, there should be a precise definition of the tool’s functionality. For a software development tool, the domain of applicability should be precisely known, and for a software verification tool, the analyses or checks it performs should be well defined.

In all cases, a tool should have sufficient safety dependability to ensure that it does not jeopardize the safety of the end product. Therefore a software development tool whose output is used without further review should be of the highest dependability level. This may be relaxed if its output is subjected to verification or if reverse engineering is used. For a software verification tool, the requirements may also be relaxed somewhat, on the grounds that its output will be closely scrutinized.

**Independent Verification and Validation**

The following is taken from the International Atomic Energy Agency, Safety Standard Series No. NS-G.1-1.
Demonstration of the correctness and safety of the system requires a variety of V&V activities. In the context of the life cycle of a computer-based system, verification is checking a product against the output of the preceding phase and validation is checking a product against higher level requirements and goals.

Validation should be performed to demonstrate that the computer system achieves its overall safety and functional requirements. There are two distinct steps of validation. The first step is the validation of the computer system requirements against the plant and system requirements. The basis for this validation against these higher level requirements and safety analyses should be explicitly identified in the validation report. The second step is the validation of the computer system implementation against the computer system requirements. Techniques and explicit validation procedures should be identified in the V&V plan.

Verification should be performed for products of the following development phases:
- Computer system design
- Software requirements
- Software design
- Software implementation

The techniques to be used to verify the software should be stated in the V&V plan, explicit procedures for the techniques should be identified and their suitability for the safety class of the system should be justified. It is expected that they will encompass a combination of techniques, including static examination of documents and dynamic execution of the implementation.

Acceptance Testing

The following is taken from DOE G 200.1-1, chapter 9.

Acceptance of a delivered software product is the ultimate objective of a software development project. Acceptance testing is used to demonstrate the software product’s compliance with the system owner’s requirements and acceptance criteria.

At the system owner’s discretion, acceptance testing may be performed by the project team, by the system owner and users with support from the project team, or by an independent V&V team. Whenever possible, users should participate in acceptance testing to ensure that the software product meets the users’ needs and expectations. All acceptance test activities should be coordinated with the system owner, user(s), operations staff, and other affected organizations.

Acceptance testing is conducted in the production environment using acceptance test data and test procedures established in the acceptance test plan. Testing is designed to determine whether the software product meets functional, performance, and operational requirements. If acceptance testing is conducted on an incremental release basis, the testing for each release should focus on the capabilities of the new release while verifying the correct operation of the requirements incorporated in the previous release.
Acceptance testing usually covers the same requirements as the system test. Acceptance testing may cover additional requirements that are unique to the operational environment. The results of each test should be recorded and included as part of the project test documentation.

Subject the test environment to strict, formal configuration control to maintain the stability of the test environment and to ensure the validity of all tests. Review the acceptance test environment, including the test procedures and their sequence, with the system owner and user before starting any tests.

Testing is complete when all tests have been executed correctly. If one or more tests fail, problems are documented, corrected, and retested. If the failure is significant, the acceptance test process may be halted until the problem is corrected.

Installation and Checkout Testing
The following is taken from the Institute of Electrical and Electronics Engineers, IEEE 1012.

In installation and checkout, the software product is installed and tested in the target environment. The installation and checkout V&V activity supports the software system installation activities. The objective of installation and checkout V&V is to verify and validate the correctness of the software installation in the target environment.

b. Describe methods for reviewing a verification and validation program.

The following is taken from DOE G 414.1-4.

The criteria review and approach document lists the following for the verification V&V program.

The V&V process and related documentation for software are defined and maintained to ensure that the software correctly performs all its intended functions, and that the software does not perform any adverse unintended function.

The criteria for the review include the following activities:

- Safety software deliverables have been verified, and validated for correct operations using reviews, inspections, assessments, observations, and testing techniques.
- Relevant abnormal conditions have been evaluated for mitigating unintended functions through testing, observations, or inspection techniques.
- Traceability of safety software requirements to software design and acceptance testing has been performed.
- New versions of the safety software are verified and validated to ensure that the safety software meets the requirements and does not perform any unintended functions.
- Verification and validation activities are performed by competent staff other than those who developed the item being verified or validated. This may overlap with the training work activity.
The approach for the review includes the following:

Review appropriate documents, such as SQA plans, review plans, walkthrough records, peer review records, desk-check records, inspection reports, test plans, test cases, test reports, system qualification plans and reports, and supplier qualification reports to determine whether

- management process exists for performing V&V and management and independent technical reviews;
- reviews and inspections of the software requirement specifications, procurement documents, software design, code modules, test results, training materials, and user documentation have been performed by staff other than those who developed the item;
- software design was performed prior to the safety software being used in operations;
- for design V&V
  - results of the safety software V&V are documented and controlled;
  - verification and validation methods include any one or a combination of design reviews, alternate calculations, and tests performed during program development; and the extent of V&V methods chosen are a function of 1) the complexity of the software; 2) the degree of standardization; 3) the similarity with previously proved software; and 4) the importance to safety
- for test V&V
  - documentation for development, factory or acceptance testing, installation, and operations testing exists;
  - documentation includes test guidelines, test procedures, test cases, including test data and expected results;
  - results documentation demonstrates successful completion of all test cases or the resolution of unsuccessful test cases and proves direct traceability between the test results and specified software design;
  - test V&V activities and their relationship with the SLC are defined;
  - software requirements and system requirements are satisfied by the execution of integration, system, and acceptance testing;
  - acceptable methods for evaluating the software test case results include 1) analysis without computer assistance, 2) other validated computer programs, 3) experiments and tests, 4) standard problems with known solutions, and 5) confirmed published data and correlations;
  - traceability exists from software requirements to design and testing, and if appropriate, to user documentation; and
  - hardware and software configurations pertaining to the V&V are specified.

c. Use an example of a safety analysis or design code to explain the following:
   - How verification primarily is the process of determining that the computational model represents the underlying mathematical model/solution.
   - How validation is the process of determining that the model accurately represents the physical phenomenon/feature being modeled (predictive capability) from the perspective of its intended use.
This is a performance-based KSA. The Qualifying Official will evaluate its completion. The following information from Sandia National Laboratories, SAND2003-3769, Verification, Validation, and Predictive Capability in Computational Engineering and Physics, may be helpful.

The fundamental strategy of verification is to identify, quantify, and reduce errors in the computational model and its numerical solution. Figure 6 depicts the verification process of comparing the numerical solution from the code in question with various types of highly accurate solutions.

*Source:* SAND2003-3769, Verification, Validation, and Predictive Capability in Computational Engineering and Physics

**Figure 6.** Verification process

Given a numerical procedure that is stable, consistent, and robust, the five primary sources of errors in computational physics solutions are: insufficient spatial discretization, insufficient temporal discretization, insufficient iterative convergence, computer round-off, and computer programming. The emphasis in verification is on identifying and quantifying errors from these various sources, and on demonstrating the stability, consistency, and robustness of the numerical scheme. Stated differently, an analytical or formal error analysis is inadequate in the verification process. Verification relies on demonstration and quantification of numerical accuracy.

The first three error sources listed in the previous paragraph are considered to be within the traditional realm of computational physics and there is extensive literature dealing with each of these topics. The fourth error source, computer round-off, is rarely dealt with in computational physics. Collectively, these four topics in verification could be referred to as solution verification or solution error assessment. The fifth error source, computer
programming, is generally considered to be in the realm of computer science or software engineering. Programming errors, which can occur, for example, in input data fields, source code programming of the numerical algorithm, output data files, compilers, and operating systems, generally are addressed using methods and tools in SQA. The identification of programming errors is usually referred to as code verification.

The fundamental strategy of validation involves identifying and quantifying the error and uncertainty in the conceptual and computational models, quantifying the numerical error in the computational solutions, estimating the experimental uncertainty, and then comparing the computational results with the experimental data. This strategy does not assume that the experimental measurements are more accurate than the computational results. The strategy only asserts that experimental measurements are the most faithful reflections of reality for the purposes of validation. Validation requires that the estimation process for error and uncertainty must occur in mathematical physics and experiments. Figure 7 depicts the validation process of comparing the computational results of the modeling and simulation process with the experimental data from various sources.

Because of the infeasibility and impracticality of conducting true validation experiments on most complex or large-scale systems, the recommended method is to use a building-block approach. This approach divides the complex engineering system of interest into at least three progressively simpler tiers: subsystems cases, benchmark cases, and unit problems. The strategy in the tiered approach is to assess how accurately the computational results compare with the experimental data at multiple degrees of physical coupling and geometric complexity. The approach is constructive in that it recognizes that there is a hierarchy of complexity in systems and simulations, and recognizes that the quantity and accuracy of information that is obtained from experiments vary radically over the range of tiers. Furthermore, this approach demonstrates that validation experiments can be conducted at many different levels of physics and system complexity. Each comparison of computational results with experimental data allows an inference of validation concerning tiers above and below the tier where the comparison is made. However, the quality of the inference depends greatly on the complexity of the tiers above and below the comparison tier.
d. **Explain the differences in the verification and validation processes for various types and applications safety software.**

The following is taken from DOE G 414.1-4.

Custom-developed software will most likely have a larger number and more detailed deliverables than would utility calculations. For level A safety software all deliverables should be reviewed using V&V methods. Additionally for level A, traceability of the requirements to the design and from requirements to test cases should be performed. For level B safety software, deliverables that include requirements, test plans and procedures, and test results should be reviewed using V&V methods.

For all level A safety software except utility calculations, acceptance testing work activities should be planned and documented; acceptance test cases and procedures, including expected results should be created; test results should be documented; and all test activity deliverables should be under configuration management. Level A utility calculations and level B and C custom-developed, configurable, acquired, and utility calculations can use a graded approach by applying less formality in the documentation. Simple check lists for acceptance test cases and procedures may be used in place of more detailed test cases and procedures. Test results
should be documented and all test activity deliverables placed under configuration management.

For level A software, continual monitoring of safety software operations based on historical failure data and results of periodic reassessment of hazards should be performed. For levels A, B, or C software, when new releases of the safety software have been developed, reviews and acceptance testing of changed documents and software should be performed.

e. **Describe the controls used to ensure that calculations performed using spreadsheets and other calculation programs are accurate. Identify the records that are maintained to document the calculation process.**

The following is taken from DOE G 414-1.4.

Modern utility calculation applications, such as spreadsheet programs, have grown dramatically in power, with a corresponding growth in risk. The addition of macro programming languages and the ability to incorporate “add-in” programs provide users with nearly the same capabilities as code developed with traditional programming tools. Utility calculation applications are installed on virtually every desktop, and user files containing algorithms and data can be easily modified by users. Calculations performed using applications such as commercial spreadsheet programs may be treated in either of two ways. In the case of relatively straightforward calculations, the calculation result may be checked and verified in the same manner as a hand calculation. For more complex or extensive calculations, where checking verification of calculation results are impractical or undesirable, the user files containing the calculation formulas, algorithms, or macros should be subject to the entire SLC process. The latter approach may also be expedient for verifying or checking calculation applications frequently.

The following is taken from Energy Facility Contractors Group, EFCOG Best Practice #65.

The results of design calculations must include not only the final output but also record of critical steps taken to reach the output, including sources of design inputs that support the final design.

Procedures that control development of design calculations should include requirements for ensuring that design input sources are derived from controlled, verified documents or standard reference sources. Documentation of the results of literature searches or other applicable background data should also be included in the design calculation. It can be inferred that documentation of the sources of “other applicable background data” applies to process specific data required for design input. Such data should be independently verified and should be representative of all expected operating conditions.

Procedures should include requirements for defining the objective and purpose of the design calculation, documenting the design inputs and assumptions that must be verified as the design proceeds, and identification of computer calculations.
Design bases, regulatory requirements and codes and standards related to the engineering calculations should be documented. Actual computations must be included in the design calculation. For computer calculations, design input and output shall be readily identifiable. Inputs to standard computer models shall be verified/validated and their use controlled and documented in accordance with an approved computer SQA program and implementing procedures. Computer calculations shall also include the computer program or software used and its version, as well as references, which validate the program/software quality and suitability for its application. For single use software not independently verified, the coding, background logic, and/or formulas used in the calculation shall be included, in addition to the results, in order to provide a basis for review and verification.

f. Describe the relationships between test procedures, test cases, expected results, test data, and actual results.

The following is taken from DOE G 200.1-1, chapter 6.

A system test plan describes the testing effort, provides the testing schedule, and defines the complete range of test cases that will be used to ensure the reliability of the software. The test cases must be complete and the expected output known before testing is started.

Test data should be varied and extensive enough to enable the verification of the operational requirements. Expected output results should be included in the test plan in the form of calculated results, screen formats, hardcopy output, predetermined procedural results, warnings, error messages and recovery.

The following is taken from DOE G 200.1-1, chapter 7.

Unit testing is driven by test cases and test data that are designed to verify software requirements, and to exercise all program functions, edits, in-bound and out-of-bound values, and error conditions identified in the program specifications. If timing is an important characteristic of the module, tests should be generated that measure time-critical paths in average and worst-case situations.

Plan and document the inputs and expected outputs for all test cases in advance of the tests. Log all test results. Analyze and correct all errors and retest the unit using the scenarios defined in the test cases. Repeat testing until all errors have been corrected. While unit testing is generally considered the responsibility of the programmer, the project manager or lead programmer should be aware of the unit test results.

The following is taken from DOE G 200.1-1, chapter 4.

The project test plan should specify the testing methodologies planned for the project, including the types of tests required, test documents, test methods, and test data collection. Each test from unit through acceptance testing is specified in terms of entrance and exit criteria and the expected level of involvement from the project team, test group, and other functional areas.
Unit and integration tests with appropriate data must be developed to exercise and validate all specified application requirements, functions, and objectives. System and acceptance tests validate that the integrated system meets the requirements.

Each type of test must use controlled computer-generated or live data as specified. The test data must be prepared to include values that will verify the functional capabilities of the software test component, identify its limitations and deficiencies (if any), exercise its capabilities, and verify that the software component performs its intended function as required.

For each type of test conducted, the test results are compared with the expected results. Discrepancies are identified and any problems resolved. Retesting is required to verify that the problem solution eliminates the problem and does not introduce new errors. The final test results are accompanied by a completed test results/error log form. This form is completed by the individual(s) responsible for testing and attached to the documents that certify the completion of each type of test.

13. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of software safety analysis.

a. Discuss the purpose and content of the following and relate the importance of each to software safety analysis, such as described in DOE G 414.1-4, Safety Software Guide for use with 10 CFR 830 Subpart A, Quality Assurance Requirements, and DOE O 414.1C, Quality Assurance and IEEE 1228, IEEE Standard for Software Safety Plans:

- Software safety plan
- Software safety requirements analysis
- Software safety design analysis
- Software safety code analysis
- Software safety test analysis
- Software safety change analysis

Software Safety Plan
The following is taken from Lawrence Livermore National Laboratory, UCRL-JC-122249.

IEEE 1228, Standard for Software Safety Plans, provides a suggested table of contents for a software safety plan, and discusses the content of each section, including software safety management, software safety analyses, and post-development procedures. Writing a safety plan, and then following the plan during software development and operation, should increase the probability of a successful and safe product.

Software Safety Requirements Analysis
The following is taken from the Institute of Electrical and Electronics Engineers, IEEE 1228.
The safety requirements analysis should define the software safety-related activities that will be carried out as part of the software requirement’s phase of development. This section of the safety software plan should specify the following:

- The types of analyses that will be performed as part of the software safety requirements analysis, and when they will occur.
- How the results of these analyses will provide the following:
  - An identification of hazards and relevant associated software requirements
  - The software safety design constraints and guidelines
  - The software safety-related test requirements and inputs to the test planning process
  - A list of required, encouraged, discouraged, and forbidden design, coding, and test techniques
- The formal review and inspection requirements for the software safety requirements analysis

**Software Safety Design Analysis**

The following is taken from the Institute of Electrical and Electronics Engineers, IEEE 1228.

The software safety design analysis defines the safety-critical activities that will be carried out as part of the software design phase of software development. This section of the software safety plan should specify the following:

- The methods by which safety-critical software design elements will be identified and classified.
- The types of analyses that will be performed on each safety-critical software design element as part of the software safety design analysis, and when they will occur.
- How the results of these analyses for each level of design will be documented. At a minimum, the documentation should specify the following:
  - Design techniques and practices covering the partitioning of the software into design elements and the effect of these techniques and practices on analyses and tests for safety
  - The relationship between system hazards and the software design elements that may affect or control these system hazards
  - The classification of each software design element according to the methods specified in the software safety plan
  - An evaluation of the software architecture for supporting the system goals for redundancy and separation of safety-critical software functions
  - An evaluation of compliance of the design with the system safety requirements
  - Tests required for the integration of subsystems and systems that will demonstrate that the software supports the system safety goals
  - Modifications to the list of required, encouraged, discouraged, and forbidden coding and test techniques
- The formal review and inspection requirements for the software safety design analysis.
Software Safety Code Analysis

The following is taken from the Institute of Electrical and Electronics Engineers, IEEE 1228.

The software safety code analysis defines the software safety-related activities that will be carried out as part of the coding phase of software development. This section of the software safety plan should specify the following:

- The analyses that will be carried out on the software code, and when they will occur.
- How the results of these analyses will be documented. At a minimum, the documentation should provide the following:
  - Identification of the specific analyses performed on each software element and the rationale for these analyses
  - Identification of the specific code analysis tools used and the rationale for the selection of those tools
  - Recommendations for design and coding changes suggested by the analyses
  - Detailed software safety-related test requirements suggested by the analyses
  - An evaluation of the compliance of the code with the safety requirement
  - Modification to the list of required, encouraged, discouraged, and forbidden coding and test techniques

- The formal review and inspection requirements for the software safety code analysis

Software Safety Test Analysis

The following is taken from the Institute of Electrical and Electronics Engineers, IEEE 1228.

The software safety test analysis defines the software safety-related activities that will be carried out as part of the software testing phase of software development. This section of the software safety plan should describe how the results of software testing will be used to show testing coverage for all software safety requirements. This section of the software safety plan should specify the following:

- The analysis that will be performed on the results of testing of software safety-critical design elements and when they will occur.
- At a minimum, the analyses should include the following:
  - The relationship between each software safety-related test and the software safety requirement that the test supports
  - Evidence that can be used to determine whether or not software safety requirements have been satisfactorily addressed by one or more software test
  - An assessment of the risk associated with the implementation as indicated by the analyses of the software tests
  - A recommendation as to whether or not adequate safety testing has been performed

- The formal review and inspection requirements for the software safety test analysis

Software Safety Change Analysis

The following is taken from the Institute of Electrical and Electronics Engineers, IEEE 1228.
The software safety change analysis should define the software safety-related activities that will be carried out in response to changes made in assumptions, specifications, requirements, design, code, equipment, test plans, environment, user documentation, and training materials. This section of the software safety plan should specify the following:

- The means for determining the impact of each change on safety
- The techniques used to determine which safety-critical software design elements are affected by changes
- The documentation to be revised to accurately reflect all software safety changes
- The analyses that must be repeated whenever the system or its environment is modified
- The extent to which regression testing is to be performed as a consequence of modifications to the system.

14. Safety software quality assurance personnel must demonstrate a familiarity level knowledge of activities that ensure that safety software will be properly maintained and will continue to operate as intended.

a. Discuss the following concepts as they relate to safety software maintenance and operation such as described in IEEE 1219, *IEEE Standard for Software Maintenance*:
   - Software maintainability
   - Maintenance planning
   - Performance monitoring
   - Preventative maintenance

*Software Maintainability*

The following is taken from Computer Practices, Using Metrics to Evaluate Software System Maintainability.

There are five methods for quantifying software maintainability from software metrics. A synopsis of the five methods is presented here:

1. Hierarchical multidimensional assessment models view software maintainability as a hierarchical structure of the source code’s attributes.
2. Polynomial regression models use regression analysis as a tool to explore the relationship between software maintainability and software metrics.
3. An aggregate complexity measure gauges software maintainability as a function of entropy.
4. Principal components analysis is a statistical technique to reduce collinearity between commonly used complexity metrics in order to identify and reduce the number of components used to construct regression models.
5. Factor analysis is another statistical technique wherein metrics are orthogonalized into unobservable underlying factors, which are then used to model system maintainability.
Tests of the models indicate that all five compute reasonably accurate maintainability scores from calculations based on simple metrics.

**Maintenance Planning**

The following is taken from the Institute of Electrical and Electronics Engineers, IEEE 1219.

Planning for maintenance may include: determining the maintenance effort, determining the current maintenance process, quantifying the maintenance effort, projecting maintenance requirements, and developing a maintenance plan.

**Determine Maintenance Effort**

The first step in the maintenance planning process is an analysis of current service levels and capabilities. This includes an analysis of the existing maintenance portfolio and the state of each system within that portfolio. At the system level, each system should be examined to determine the following:

- Age since being placed in production
- Number and type of changes during life
- Usefulness of the system
- Types and number of requests received for changes
- Quality and timeliness of documentation
- Any existing performance statistics

Descriptions at the portfolio level can assist in describing the overall effort and needs of the maintenance area. This includes the amount and kinds of functional system overlap and gaps within the portfolio architecture.

The reviews of the maintenance staff and the maintenance procedures are also necessary to determine the overall maintenance effort. The analysis at this stage is used to gather those measures needed to determine the following:

- The number of maintainers, their job descriptions, and their actual jobs
- The experience level of the maintenance staff
- The rate of turnover and possible reasons for leaving
- Current written maintenance methods at the systems and program level
- Actual methods used by programming staff
- Tools used to support the maintenance process and how they are used

Information at this stage is used to define the baseline for the maintenance organization and provide a means of assessing the necessary changes.

**Determine Current Maintenance Process**

The maintenance process is a natural outgrowth of many of the baseline measures. Once those measures have been collected, the actual process needs to be determined. In some organizations, the process is tailored to the type of maintenance being performed and can be divided in several different ways. This can include different processes for corrections vs. enhancements, small changes vs. large changes, etc. It is helpful to classify the maintenance approaches used before defining the processes.
Each process will then be described by a series of events. In general, the flow of work is described from receipt of a request to its implementation and delivery.

Quantify Maintenance Effort
Each step in the process needs to be described numerically in terms of volume or time. These numbers can then be used as a basis to determine the actual performance of the maintenance organization.

Project Maintenance Requirements
At this stage, the maintenance process needs to be coupled to the business environment. A review of future expectations should be completed and may include the following:

- Expected external or regulatory changes to the system
- Expected internal changes to support new requirements
- Wish-list of new functions and features
- Expected upgrades for performance, adaptability, connectivity, etc.
- New lines of business that need to be supported
- New technologies that need to be incorporated

These need to be quantified to determine the future maintenance load for the organization.

Develop Maintenance Plan
The information collected will provide a basis for a new maintenance plan. The plan should cover the following main areas:

- Maintenance process
- Organization
- Resource allocations
- Performance tracking

Each of these issues are addressed and embedded in the final maintenance plan. The actual process should be described in terms of its scope, the sequence of the process, and the control of the process.

Process Scope
The plan needs to define the boundaries of the maintenance process. The process begins at some point and will end with some action. In addition, the difference between maintenance and development should be addressed at this point. Is an enhancement considered to be a new development or maintenance? At what point does a newly developed system enter the maintenance process.

Another issue that should be defined within the scope is whether and how the maintenance process will be categorized. Will there be differences between reporting and other types of maintenance? Will adaptations and enhancements be considered within the same process or will they be handled differently?

Process Sequence
The overall flow of work needs to be described. This should include the following:
- Entry into automated SCM and project management systems
- Descriptions of each process step and its interfaces
- The data flow between processes

**Performance Monitoring**

The following is taken from DOE Quality Report SQAS97-001.

The following paragraphs present recommendations for implementing a set of four core measures to support software projects. Methods for defining and reporting results are provided for each measure. These methods are supported with reasons for using the measures and recommendations are included for making the measures effective. The recommended core software measures are listed in table 4 along with examples of measure units and the characteristics they address. Software development projects within any organization should use these measures whether acquiring, developing, using or maintaining software systems.

**Table 4. Measures for initial implementation**

<table>
<thead>
<tr>
<th>Type of Measure</th>
<th>Examples of Measure Units</th>
<th>Characteristic Addressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Counts of physical source lines of code</td>
<td>Size, progress, reuse</td>
</tr>
<tr>
<td></td>
<td>Function points</td>
<td></td>
</tr>
<tr>
<td>Effort</td>
<td>Counts of staff hours expended</td>
<td>Effort, cost, resource use, rework</td>
</tr>
<tr>
<td></td>
<td>Counts of staff hours to correct problems and defects</td>
<td></td>
</tr>
<tr>
<td>Progress to schedule</td>
<td>Calendar dates</td>
<td>Schedule, progress</td>
</tr>
<tr>
<td>Defects</td>
<td>Counts of software problems and defects</td>
<td>Quality, acceptability for delivery, improvement trends, customer satisfaction</td>
</tr>
</tbody>
</table>

*Source:* DOE Quality Report SQAS97-001

Although other measurements also capture attributes of software resources, products and processes, the measures listed above are practical, produce meaningful information, and can be defined to promote consistent use. In that regard, the Software Engineering Institute has developed three reports that are useful in defining the methods that are to be used in collecting software measurements. In addition, these reports allow software project management to state what each basic measure includes and excludes. The reports are

- Software Size Measurement: A Framework for Counting Source Statements
- Software Effort and Schedule Measurement: A Framework for Counting Staff Hours and Reporting Schedule Information
- Software Quality Measurement: A Framework for Counting Problems and Defects
Because experienced software managers are rarely satisfied with a single number, each measure requires collection of multiple data items to define the attribute mapping. For example, problems and defects usually are classified according to such attributes as process task activity, status, type, severity, and priority. Effort can be classified by labor class and type of process activity performed. Schedules are defined by process activity, dates and completion criteria. Size measures might be aggregated according to programming language, development status, and production method. To be of value, estimated and measured values must be collected at regular intervals.

What may first appear to be just a few measures is actually much more. It will be a significant accomplishment to implement a uniform collection and use of these measures across a single organization, much less the many organizations within any given site. Neither the difficulty nor the value of this task should be underestimated.

Size
Size of software product is dependent upon the attributes of product length, functionality, and complexity. Some of the more popular and effective measures of the length part of software size are physical source lines of code and a logical source statement. Function points and counts of logical functions or CSUs might be the measure of the functionality part of software size.

Effort
Reliable measures for effort are prerequisites to dependable measures of software cost. By tracking the human resources assigned to individual tasks and activities, effort measures also provide the principal means for managing and controlling costs and schedules.

The measurement of effort and schedule is recommended. The definition of effort in the IEEE’s draft Standard for Software Productivity Measurements is the staff hour unit.

Although other units for measuring and reporting effort data could be used, such as staff months and staff weeks, the use of staff hours is preferable.

Progress in Schedule
Schedule and progress are primary project management concerns. For many projects timely delivery is often as important as functionality or quality in determining the ultimate value of a software product. Moreover, project management can become especially complicated when delivery dates are determined by external constraints rather than by the inherent size and complexity of the software product. Overly ambitious, unrealistic schedules often result.

Because schedule is a key concern, it is important for managers to monitor adherence to intermediate milestone dates. Early schedule slips often foreshadow future problems. It is also important to have objective and timely measures of progress that accurately indicate status and that can be used to estimate completion dates for future milestones.
Cost estimators and cost model developers are also very interested in schedules. Project duration is a key parameter when developing or calibrating cost models. Model developers and estimators must understand what activities the duration includes and excludes.

Defects
Determining what a customer or user views as true software quality can be elusive. Whatever the criteria, it is clear that the number of problems and defects associated with a software product varies inversely with perceived quality. Counts of software problems and defects are among the few direct measures for software processes and products. These counts allow qualitative description of trends in detection and repair activities. They also allow the tracking of progress in identifying and fixing process and product imperfections. The counts and rework time should be used to help determine when products are ready for delivery to customers, identify the operational impact of software defects on the customer, and provide fundamental data for process and product improvement. In addition, problem and defect measures are the basis for quantifying other software quality attributes such as reliability, correctness, completeness, efficiency, and usability.

Defect correction is a significant cost in most software development and maintenance environments. The number of problems and defects associated with a product contribute directly to this cost. Counting problems and defects can aid understanding of where and how they occur and provide insight into methods for early detection, prevention, and prediction. Counting problems and defects also can directly help track project progress, identify process inefficiencies, and forecast obstacles that will jeopardize schedule commitments.

Preventive Maintenance
The following is taken from Tufts University, Software Maintenance as Part of the Software Life Cycle.

Preventive maintenance concerns activities aim at increasing the system’s maintainability, such as updating documentation, adding comments, and improving the modular structure of the system. The long-term effect of corrective, adaptive and perfective changes increases the system’s complexity. As a large program is continuously changed, its complexity, which reflects deteriorating structure, increases unless work is done to maintain or reduce it. This work is known as preventive change. The change is usually initiated from within the maintenance organization with the intention of making programs easier to understand and hence facilitating future maintenance work. Examples of preventive change include restructuring and optimizing code and updating documentation.

15. Safety software quality assurance personnel must demonstrate a working level knowledge of software configuration management.

a. Discuss the following concepts as they relate to safety software configuration management such as described in IEEE 828, IEEE Standard for Software Configuration Management Plans and explain how each is applied:
   • Software configuration management plan
   • Configuration identification
• Configuration change control
• Configuration status accounting
• Configuration audits and reviews
• Subcontractor and vendor control
• Software configuration management tools, including source code control tools

The information for this KSA is taken from IEEE 828.

Software Configuration Management Plan

The SCM plan documents what SCM activities are to be done, how they are to be done, who is responsible for doing specific activities, when they are to happen, and what resources are required. It can address SCM activities over any portion of a software product’s life cycle.

Software configuration management planning information should be partitioned into the following classes:
• Introduction
• SCM management
• SCM activities
• SCM schedules
• SCM resources
• SCM plan maintenance

Introduction
Introduction information provides a simplified overview of the SCM activities so that those approving, those performing, and those interacting with SCM can obtain a clear understanding of the plan. The introduction should include four topics: the purpose of the plan, the scope, the definition of key terms, and references.

SCM Management
SCM management information describes the allocation of responsibilities and authorities for SCM activities, and their management, to organizations and individuals within the project structure. SCM management information should include four topics: the project organizations within which SCM is to apply, the SCM responsibilities of these organizations, reference to the SCM policies and directives that apply to this project, and the management of the SCM process.

SCM Activities
SCM activities information identifies all functions and tasks required to manage the configuration of the software system as specified in the scope of the plan. Technical and managerial SCM activities should be identified. General project activities that have SCM implications should be described from the SCM perspective. SCM activities are traditionally grouped into five functions: configuration identification, configuration control, configuration status accounting, configuration evaluations, and reviews, and release management and delivery.
SCM Schedules
SCM schedule information establishes the sequence and coordination for the identified SCM activities and for all events affecting the plan’s implementation.

The plan should state the sequence and dependencies among all SCM activities and the relationship of key SCM activities to project milestones or events. The schedule should cover the duration of the plan and contain all major milestones of the project related to SCM activities. Milestones should include establishment of a configuration baseline, implementation of change control procedures, and the start and completion dates for a configuration audit.

Schedule information should be expressed as absolute dates, as dates relative to either SCM or project milestones, or as a simple sequence of events. Graphic representation can be particularly appropriate for conveying this information.

SCM Resources
SCM resource information identifies the environment, infrastructure, software tools, techniques, equipment, personnel, and training necessary for the implementation of the specified SCM activities.

SCM can be performed, within an overall environment or infrastructure, by a combination of software tools and manual procedures. Tools can be SCM-specific or embedded in general project aids; they can be standard organizational resources or ones specially acquired or built for this project. Tools can be applied to library structure and access control, documentation development and tracking; code control; baseline system generations; change processing, communication and authorization; change/problem tracking and status reporting; archiving, retention, and retrieval of controlled items; or the SCM planning process itself.

SCM Plan Maintenance
SCM plan maintenance information identifies the activities and responsibilities necessary to ensure continued SCM planning during the life cycle of the project. The plan should include a history of changes made to the plan and state the following:

- Who is responsible for monitoring the plan
- How frequently updates are to be performed
- How changes to the plan are to be evaluated and approved
- How changes to the plan are to be made and communicated

The plan should be reviewed at the start of each project software phase, changed accordingly, and approved and distributed to the project team.

If the plan has been constructed with detailed procedures documented elsewhere in appendixes or references, different maintenance mechanisms for those procedures may be appropriate.
**Configuration Identification**

Configuration identification activities should identify, name, and describe the documented physical and functional characteristics of the code, specifications, design, and data elements to be controlled for the project. The documents are acquired for configuration control. Controlled items may be intermediate and final outputs. These items include outputs of the development process and elements of the support environment.

The plan should identify the project configuration items and their structures at each project control point. The plan should state how each configuration item and its versions are to be uniquely named and describe the activities performed to define, track, store, and retrieve configuration items.

**Configuration Change Control**

Configuration control activities request, evaluate, approve or disapprove, and implement changes to baselined configuration items. Changes encompass error correction and enhancement. The degree of formality necessary for the change process depends on the project baseline affected and on the impact of the change within the configuration structure. Configuration control activities also apply to the processing of requests for deviations and waivers from the provisions of specifications or acquirer-supplier contractors.

The plan should describe the change controls imposed on the baselined configuration items. The plan should define the following sequence of specific steps:

- Identification and documentation of the need for a change
- Analysis and evaluation of a change request
- Approval or disapproval of a request
- Verification, implementation, and release of a change

The plan should identify the records to be used for tracking and documenting this sequence of steps for each change. Any variations in handling changes based on the origin of the request shall be explicitly documented.

**Configuration Status Accounting**

Configuration status accounting activities record and report the status of project configuration items. The plan should include information on the following:

- What data elements and SCM metrics are to be tracked and reported for baselines and changes
- What types of status accounting reports are to be generated and their frequency
- How information is to be collected, stored, processed, reported, and protected from loss
- How access to the status data is to be controlled

If an automated system is used for any status accounting activity, its function should be described or referenced. Status accounting records and reports should be available that provide the current status and history of controlled items. The following minimum data elements should be tracked and reported for each configuration item: its approved versions,
the status of requested changes, and the implementation status of approved changes. The level of detail and specific data required may vary according to the information needs of the project and the customer.

_Configuration Audits and Reviews_

Configuration evaluation consists of audits that determine the extent to which the actual configuration item reflects the required physical and functional characteristics. Configuration reviews are a management mechanism to evaluate a baseline.

The plan should identify the configuration audits and reviews to be held for the project. At a minimum, a configuration audit should be performed on a configuration item prior to its release.

For each planned configuration audit or review, the plan should define the following:

- Its objective
- The configuration items under audit or review
- The schedule of audit or review tasks
- The procedures for conducting the audit or review
- The participants by job title
- Documentation required to be available for review or to support the audit or review
- The procedure for recording any deficiencies and reporting corrective actions
- The approval criteria and the specific actions to occur upon approval

_Subcontractor and Vendor Control_

Subcontractor/vendor control activities incorporate items developed outside the project environment into the project configuration items. Included are software developed by contract and software acquired in its finished form. Special attention should be directed to these SCM activities due to the added organizational and legal relationships.

For subcontracted and acquired software, the plan should define the activities to incorporate the externally developed items into the project configuration items and to coordinate changes to these items with their development organizations.

For subcontracted software, the plan should describe the following:

- What SCM requirements, including an SCM plan, are to be part of the subcontractor’s agreement
- How the subcontractor will be monitored for compliance
- What configuration evaluations and reviews of subcontractor items will be held
- How external code, documentation, and data will be tested, verified, accepted, and merged with the project software
- How proprietary items will be handled for security of information and traceability of ownership
- How changes are to be processed, including the subcontractor’s participation
Acquired software items should be placed under SCM. The plan should describe how the software will be received, tested, and placed under SCM; how changes to the supplier’s software are to be processed; and whether and how the supplier will participate in the project’s change management process. Acquired software can come from a vendor, a subcontractor, a customer, another project, or other source.

**Software Configuration Management Tools, Including Source Code Control Tools**

The following is taken from *DOE Software Configuration Management: A Practical Guide*.

The SCM automated tools used for a project and described in the SCM plan need to be compatible with the software engineering environment in which the development or maintenance is to occur. SCM tools offer a wide range of capabilities and choices have to be made as to the tool set most useful for supporting the engineering and management environment.

There are many different ways of evaluating SCM tools. One way is to categorize the tools according to characteristics of their products, such as: a filing system, a database management system, and an independent knowledge-based system. Another way is to examine the functions that the tools perform, such as: clerical support, testing and management support, and transformation support. A third way of categorizing the SCM tools is by how they are integrated into the software engineering environment on the project.

The automated tools described in the following paragraphs are classified into broad categories in terms of the level of automation they provide to the programming environment on a project.

**Basic Tool Set**

The basic tool set is compatible with a programming environment that is relatively unsophisticated. The tools control the information on hard copy regarding a program product. These tools provide a capability that distinguishes between controlled and uncontrolled units or components. The tools simplify and minimize the complexity, time, and methods needed to generate a given baseline. The basic tool set includes:

- basic database management systems
- report generators
- means for maintaining separate dynamic and controlled libraries
- file system for managing the check-in and check-out of units, for controlling compilations, and capturing the resulting products

**Advanced Tool Set**

The advanced tool set provides a capability for an SCM group to perform more efficiently on larger, more complex software engineering projects. These tools provide a programming environment that has more computing resources available. It provides the means of efficiently managing information about the units or components and associated data items. It also has rudimentary capabilities for managing the configurations of the product and providing for more effective control of the libraries. The advanced tool set includes
items in the basic tool set
source code control programs that will maintain version and revision history
tools for comparing programs for identifying changes
tools for building or generating executable code
a documentation system to enter and maintain the specifications and associated user
documentation files
a system/software change request/authorization tracking system that makes requests
for changes machine-readable
capability to manage concurrent development efforts

Online Tool Set
The online tool set requires an interactive programming environment that is available to the
project. It also provides an organization with the minimal SCM capabilities needed to support
the typical interactive programming environment currently available in industry. These tools
provide online access to the programming database and the resources necessary for using the
tools. The online tool set includes
generic tools of the advanced tool set integrated so they work from a common
database
a software change request tracking and control system that brings generation, review,
and approval of changes online
report generators working online with the common database, and a software change
request tracking system that enables the SCM group to generate responses to online
queries of a general nature

Integrated Tool Set
The integrated tool set integrates the SCM functions with the software engineering
environment so that the SCM functions are transparent to the engineer. The software
engineer becomes aware of the SCM functions only when he/she attempts to perform a
function or operation that has not been authorized. An integrated tool set includes
online SCM tools covering all functions
an integrated engineering database with SCM commands built into the online
engineering commands commonly used in designing and developing programs
the integration of the SCM commands with online management commands for
building and promoting units and components

Tool Selection
Software configuration management tools should be carefully selected to match working
practices and ideology. Selection of SCM tools should be based on the following features:
Cross-platform support
Developer empowerment
Match to existing work practices
Tool integration
Ease of installation and use
Code visibility outside tool
Supplier support before and after delivery
- Market position of supplier and product
- Overall cost

While the selection of tools will be influenced by the functional requirements, tools are characterized by a series of additional attributes that reflect software engineering practices.

b. Review and evaluate an unreviewed safety question (USQ) determination associated with software including any proposed change or potential inadequacy.

This is a performance-based KSA. The Qualifying Official will evaluate its completion.

c. Discuss the process of problem reporting and corrective actions and its implementation related to software configuration management.

The following is taken from *DOE Software Configuration Management: A Practical Guide*.

Once configuration items have been identified and baselined, configuration control provides a system to initiate, review, approve, and implement proposed changes. Each engineering change or problem report that is initiated against a formally identified configuration item is evaluated to determine its necessity and impact. Approved changes are implemented, testing is performed to verify that the change was correctly implemented and that no unexpected side effects have occurred as a result of the change, and the affected documentation is updated and reviewed.

The discipline of status accounting collects information about all configuration management activities. The status account should be capable of providing data on the status and history of any configuration item. The information maintained in status accounting should enable the rebuild of any previous baseline.

Audits and reviews are conducted to verify that the software product matches the configuration item descriptions in the specifications and documents, and that the package being reviewed is complete. Any anomalies found during audits should be corrected and the root cause of the problem identified and corrected to ensure that the problem does not resurface. Generally, there should be a physical configuration audit (PCA) and the functional configuration audit (FCA) of configuration items prior to the release of a software product baseline or an updated version of a product baseline. The PCA is performed to determine if all items identified as being part of the configuration are present in the product baseline. The audit must also establish that the correct version and revision of each part are included in the product baseline and that they correspond to information contained in the baseline’s configuration status report. The FCA is performed on each software configuration item to determine that it satisfies the functions defined in the specifications or contracts for which it was developed.
16. Safety software quality assurance personnel must demonstrate the ability to perform an assessment of safety software quality using appropriate DOE directives and standards and industry standards.

a. Participate in and document one or more SQA assessments that demonstrate the ability to adequately assess SQA work activities selected from the following as described in DOE-O 414.1C, Quality Assurance and DOE G 414.1-4, Safety Software Guide for use with 10 CFR 830 Subpart A, Quality Assurance Requirements, and DOE O 414.1C, Quality Assurance:

- Software project management and quality planning
- Software risk management
- Software configuration management
- Procurement and supplier management
- Software requirements identification and management
- Software design and implementation
- Software safety
- Verification and validation
- Problem reporting and corrective action
- Training of personnel in the design, development, testing, use, and evaluation of safety software

This is a performance-based KSA. The Qualifying Official will evaluate its completion. The following information taken from DOE G 414.1-4 may be helpful.

*Software Project Management and Quality Planning*

As with any system, project management and quality planning are key elements to establishing the foundation to ensure a quality product that meets project goals. For software, project management starts with the system-level project management and quality planning. Software specific tasks should be identified and either included within the overall system planning or in separate software planning documents.

These tasks may be documented in an SPMP, an SQA plan, an SDP, or similar documents. They also may be embedded in the overall system-level planning documents. Typically these plans are the controlling documents that define and guide the processes necessary to satisfy project requirements, including the software quality requirements. These plans are initiated early in the project life cycle and are maintained throughout the life of the project.

The software project management and quality planning should include identifying all tasks associated with the software development and procurement, including procurement of services, estimate of the duration of the tasks, resources allocated to the tasks, and any dependencies. The planning should include a description of the tasks and any relevant information.
Software quality and software development planning identifies and guides the software phases and any grading of the SQA and software development activities performed during software development or maintenance. The software quality and software engineering activities and rigor of implementation will be dependent on the identified grading level of safety software and the ability of DOE or its contractors to build quality in and assess the quality of the safety software. Because the SQAP and SDP are overall basic quality and software engineering, some quality activities, such as SCM, risk management, problem reporting, and corrective actions, and V&V, including software reviews and testing, may be further detailed in separate plans.

Software project management and quality planning fully apply to custom-developed and configurable software types for level A and level B safety software. For level A and level B acquired and utility calculation and all level C software applications, software project management and quality planning tasks can be graded. This grading should include the identification and tracking of all significant software tasks. Where instances of the SLC may include little or no software development activities, the software project and quality planning will most likely be part of the overall system-level project or facility planning. This work activity does not apply to commercial design and analysis software because the project management and quality planning activities associated with commercial design and analysis software are performed by the service supplier. DOE controls the SQA activities of that software through procurement agreements and specifications.

**Software Risk Management**

Software risk management provides a disciplined environment for proactive decision-making to continuously assess what can go wrong, determine what risks are important to address, and implement actions to address those risks. Because risk management is such a fundamental tool for project management, it is an integral part of software project management. Although sometimes associated with safety analysis of potential failures, software risk management focuses on the risks to the successful completion of the software project.

Risk assessment and risk control are two fundamental activities required for project success. Risk assessment addresses identification of the potential risks, analysis of those risks, and prioritizing the risks to ensure that the necessary resources will be available to mitigate the risks. Risk control addresses risk tracking and resolution of the risks. Identification, tracking, and management of the risks throughout all phases of the project’s life cycle should include special emphasis on tracking the risks associated with costs, resources, schedules, and technical aspects of the project.

Examples of potential software risks for the safety software application might include

- incomplete or volatile software requirements
- specification of incorrect or overly simplified algorithms or algorithms that will be very difficult to address within safety software
- hardware constraints that limit the design
- potential performance issues with the design
- a design that is based on 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
- use of a programming language with only minimal experience using the language
- 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

**Software Configuration Management**

Software configuration management activities identify all functions and tasks required to manage the configuration of the software system, including software engineering items, establishing the configuration baselines to be controlled, and software configuration change control process. The following four areas of SCM should be addressed when performing configuration management: 1) configuration identification, 2) configuration control, 3) configuration status accounting, and 4) configuration audits and reviews.

The methods to control, uniquely identify, describe, and document the configuration of each version or update of software and its related documentation should be documented. This documentation may be included in an SCM plan or its equivalent. Such documentation should include criteria for configuration identification, change control, configuration status accounting, and configuration reviews and audits.

**Procurement and Supplier Management**

Most software projects will have procurement activities that require interactions with suppliers regardless of whether the software is level A, B, or C. Procurement activities may be as basic as the purchase of compilers or other development tools for custom-developed software or as complicated as procuring a complete safety system software control system.

There are a variety of approaches for software procurement and supplier management based on
- the level of control DOE or its contractors have on the quality of the software or software service being procured and
- the complexity of the software

Procurement documentation should include the technical and quality requirements for the safety software. Some of the specifications that should be included are
- specifications for software features, including requirements for safety, security, functions, and performance;
- process steps used in developing and validating the software, including any documentation to be delivered;
- requirements for supplier notification of defects, new releases, or other issues, that impact the operation;
• mechanisms for the users of the software to report defects and request assistance in operating the software.

These requirements should be assessed for completeness and to ensure the quality of the software being purchased. There are four major approaches for this assessment:
1. Performing an assessment of the supplier
2. Requiring the supplier to provide a self-declaration that the safety software meets the intended quality
3. Accepting the safety software based on key characteristics
4. Verifying the supplier has obtained a certification or accreditation of the software product quality or software quality program from a third party

Software Requirements Identification and Management

Safety system requirements provide the foundation for the requirements to be implemented in the software. These systems requirements should be translated into requirements specific for the software. The identified software requirements may be documented in system-level requirements documents, SRSs, procurement contracts, and/or other acquired software agreements. These requirements should identify functional; performance; security, including user access control, interface and safety requirements; and installation considerations and design constraints where appropriate. The requirements should be complete, correct, consistent, clear, verifiable, and feasible.

Once the 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 SLC.

Software Design and Implementation

During software design and implementation the software is developed, documented, reviewed, and controlled. The software design elements should identify the operating system, function, interfaces, performance requirements, installation considerations, design inputs, and design constraints. The software design should be complete and sufficient to meet the software requirements. The design activities and documentation should be adequate to fully describe how the software will interface with other system components, and how the software will function internally. Data structure requirements and layouts may be necessary to fully understand the internal operations of the software.

Custom-developed software will require more formality in the documentation and review of the design than configurable or utility calculations. Simple process flows, relationships between data elements, interfaces with external components, and basic database table structures may be all that are needed for configurable or utility calculations, whereas for custom-developed software, complete functional and logical designs of the software components, the input and output data, and pseudo code may be required to fully understand the safety software design. The software design description may be combined with the documentation of the software requirements or software source code.
During implementation, static analysis, cleanroom inspections, and reviews are common techniques to ensure the implementation remains consistent with the design and does not add complexity or functions that could decrease the safe operation of the software. Many tools exist to evaluate the complexity and other attributes of the source code design structure.

The software developed should perform unit testing prior to system level V&V techniques, including acceptance testing. Developer testing can be very structured and formal, using automated tools or less formal methods. In addition to unit testing, functional, structural, timing, stress, security, and human-factors testing are useful testing methods.

**Software Safety**

The development of software applications requires identification of hazards that have the potential for defeating a safety function and the implementation of design strategies to eliminate or mitigate those hazards. It is recommended that the software safety process address the mitigation strategy for the components that have potential safety consequences if a fault occurs, whereas the software design and implementation process addresses the architecture of the safety software application.

Software is only one component of the overall safety system. It may be embedded in an instrumentation and control system, it may be a custom control system for hardware components, or it may be stand-alone software used in safety management or support decisions. In any of these or other applications of software important to safety, analysis of the software application occurs first at the system level. The analysis should then be performed at the software component level to ensure adequate safeguards are provided to eliminate or mitigate the potential occurrence of a software defect that could cause a system failure.

Methods to mitigate the consequences of software failures should be an integral part of the software design. Specific software analysis and design methods for ensuring that safety functions are well thought out and addressed properly should be performed throughout the software development and operations life cycles. These methods include dynamic and static analyses.

**Verification and Validation**

Verification and validation is the largest area within the SQA activities. Verification is performed throughout the life cycle of the safety software. Validation activities are performed at the end of the software development or acquisition processes to ensure the software meets the intended requirements. Verification and validation activities should be performed by competent staff other than those who developed the item being verified or validated. Verification and validation activities include reviews, inspections, assessments, observations, and testing.

Reviews and inspections of software deliverables requirement specifications, procurement documents, software design, code modules, test results, training materials, user documentation, and processes that guide the software development activities should be performed.
Problem Reporting and Corrective Action

Coupled with the configuration management of the software system, the problem reporting and corrective action process should address the appropriate requirements of the QAP corrective action system. The reporting and corrective action system will cover methods for documenting, evaluating and correcting software problems, and evaluation process for determining whether a reported problem is indeed a defect or an error, and the roles and responsibilities for disposition of the problem reports, including notification to the originator of the results of the evaluation. If the noted problem is indeed an error, the problem reporting and corrective action system should correlate the error with the appropriate software engineering elements; identify the potential impacts and risks to past, present, and future developmental and operational activities; and support the development of mitigation strategies. After an error has been noted, all users should be apprised to ascertain any impacts on safety basis decisions.

Maintaining a robust problem reporting and corrective action process is obviously vital to maintaining a reliable and vital safety software system. 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.

Training of Personnel in the Design, Development, Testing, use, and Evaluation of Safety Software

Training personnel in designing, developing, testing, evaluating, or using the safety software application is critical for minimizing the consequences of software failure. Although other SQA work activities may indicate that the software satisfies its operational objective, improper or invalid use of the software may negate the safety mitigation strategies included within the software.

Training may be necessary for the analysis, development and test teams, application users, and operations staff. The analyst and developers may need training in fault-tolerant methodologies, safety design methodologies, user interface design issues, testing methodologies, or configuration management to ensure delivery of a robust software application. Meanwhile, the software application users and operations staff may need training specific to the software to ensure that proper data are entered, that proper options and menus are selected, and that the results of the software can be interpreted correctly. A trained and knowledgeable staff is essential to assess and evaluate the SQA requirements to ensure the proper levels of quality and safety exists in the software.

b. Safety software quality assurance personnel must demonstrate through an assessment the ability to evaluate safety software in relation to the safety basis, and explain how any changes that may affect the safety software function are controlled. The following are examples of activities that may demonstrate this ability:

- Identify and discuss the safety function of safety software related to the prevention or mitigation of hazards or accidents identified in the safety basis
documentation. Additionally, identify the facility equipment and/or system by walking down with a cognizant system engineer.

- Review the function of safety software that controls a safety function as defined in a technical safety requirement (TSR) or an administrative control, including the technical basis for the control.
- Review permanent or temporary changes made to a safety system digital instrumentation and control (I&C) that involve changes to safety software configuration and implementation; evaluate how those changes were processed through the facility’s USQ procedures; review how any changes made to the safety software were documented and review verified; and how any changes to the safety basis documentation or facility procedures were implemented. Summarize the review activity along with any observations and insights gained.
- Identify some of the major input parameters of a safety software and summarize how they are being controlled to prevent any undesired changes.

This is a performance-based KSA. The Qualifying Official will evaluate its completion.
Selected Bibliography and Suggested Reading

**Code of Federal Regulations (CFR)**


Answers.com, Engineering, Operating Systems.


eHow, How to Know the Fundamentals of Program Structure.

Energy Facility Contractors Group, EFCOG Best Practice #65, Engineering Calculations.


**Institute of Electrical and Electronics Engineers**


Novedge, Maya Help, Embedded Comments.

Project Management Tips—Four Cases for Early Requirements Verification.


U.S. Department of Energy Directives (Guides, Manuals, Orders, and Policies)

U.S. Department of Energy Handbooks and Standards

U.S. Department of Energy Other References
Pacific Northwest National Laboratory, New DOE Software Quality Assurance Requirements and Implications for Meteorology and Consequence Assessment Software, What is Safety Software?
Path Forward to Address Gaps in Toolbox Code Gap Analysis Reports.
Wiegers, Karl E., Seven Truths about Peer Reviews. 2002.