ANL/NSE-23/13

Software Quality Assurance Plan

Argonne Reactor Code Software

Nuclear Science and Engineering Division

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Lemont, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

ANL/NSE-23/13

Software Quality Assurance Plan

Argonne Reactor Code Software

Prepared by:

Kalin Kiesling and M. A. Smith

Nuclear Engineering Division, Argonne National Laboratory

March 01 2023

Prepared by: Kalin Kiesling Date: 3/30/2023

Kalin Kiesling

Software Development Team Member

Reviewed by: John B. Woodford Digitally signed by John B. Woodford Date: 2023.03.31 14:09:37 -05'00' Date:

John Woodford

NSE QAR

Approved by: MICHEAL SMITH (Affiliate) Digitally signed by MICHEAL SMITH (Affiliate) Date: 2023.04.19 11:44:04 -05'00' Date:

Micheal A Smith

Software Manager

TABLE OF CONTENTS

LIST OF TABLES

1 Introduction

1.1 Project Background

The Argonne Reactor Code (ARC) software package consists of 13 primary and many secondary pieces of software that are connected through interface files. The ARC software suite supports users in their fast reactor design goals by providing neutronic, thermal-hydraulic, and structural analysis capabilities. Specifically, the major software components of ARC that are in scope for this Software Quality Assurance Plan (SQAP) are: DIF3D [2], REBUS [3], RCT [6], PERSENT [4], VARI3D [4], GAMSOR [5], SE2ANL [7], SE2RCT [7], NUBOW-3D [8], DIF3D-K [9] and DASSH [10]. Figure 1.1 shows the typical design process workflow for the fuel cycle analysis portion where the cited codes are named in orange boxes. Figure 1.2 shows the connected software that is typically used for the thermal-hydraulics, mechanics, and safety analysis portion of the design process. Note that in Figure 1.1 and Figure 1.2, ETOE [1], MC^2-3 [1], and SAS4A [11] are listed as connected components. They are included in other software quality assurance (SQA) programs and thus not covered as part of this SQA program.

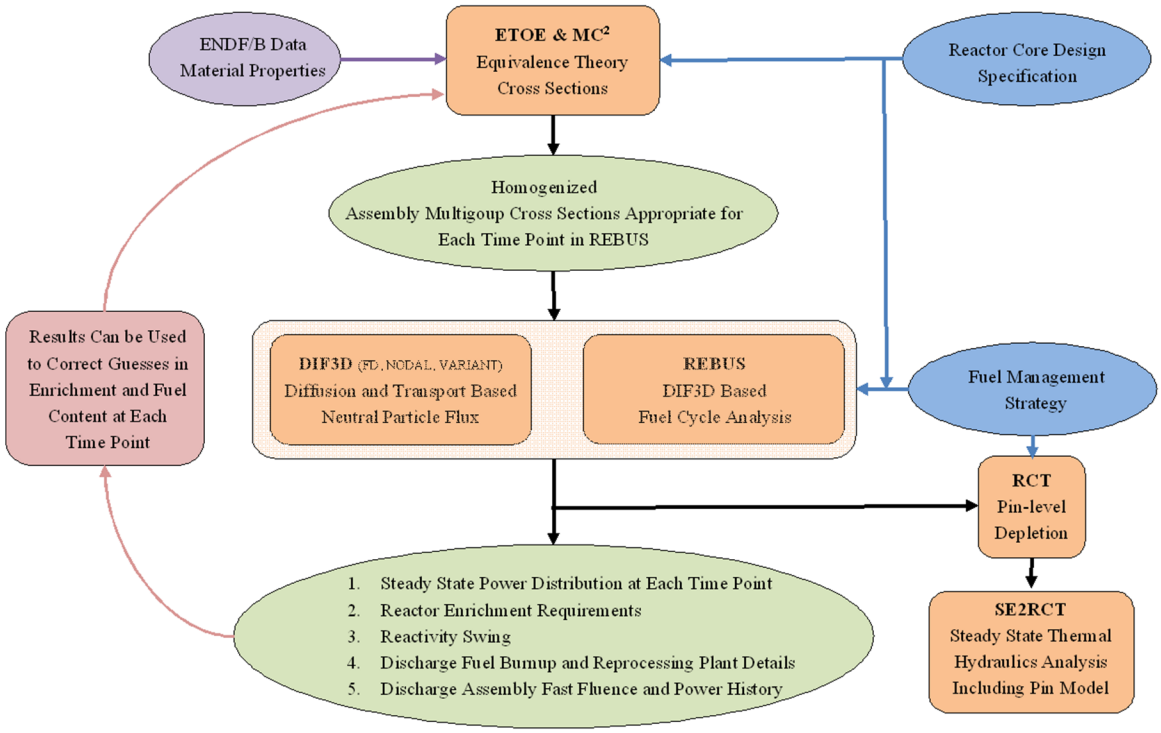

Figure 1.1: ARC Software Connections for Fast Spectrum Fuel Cycle Analysis

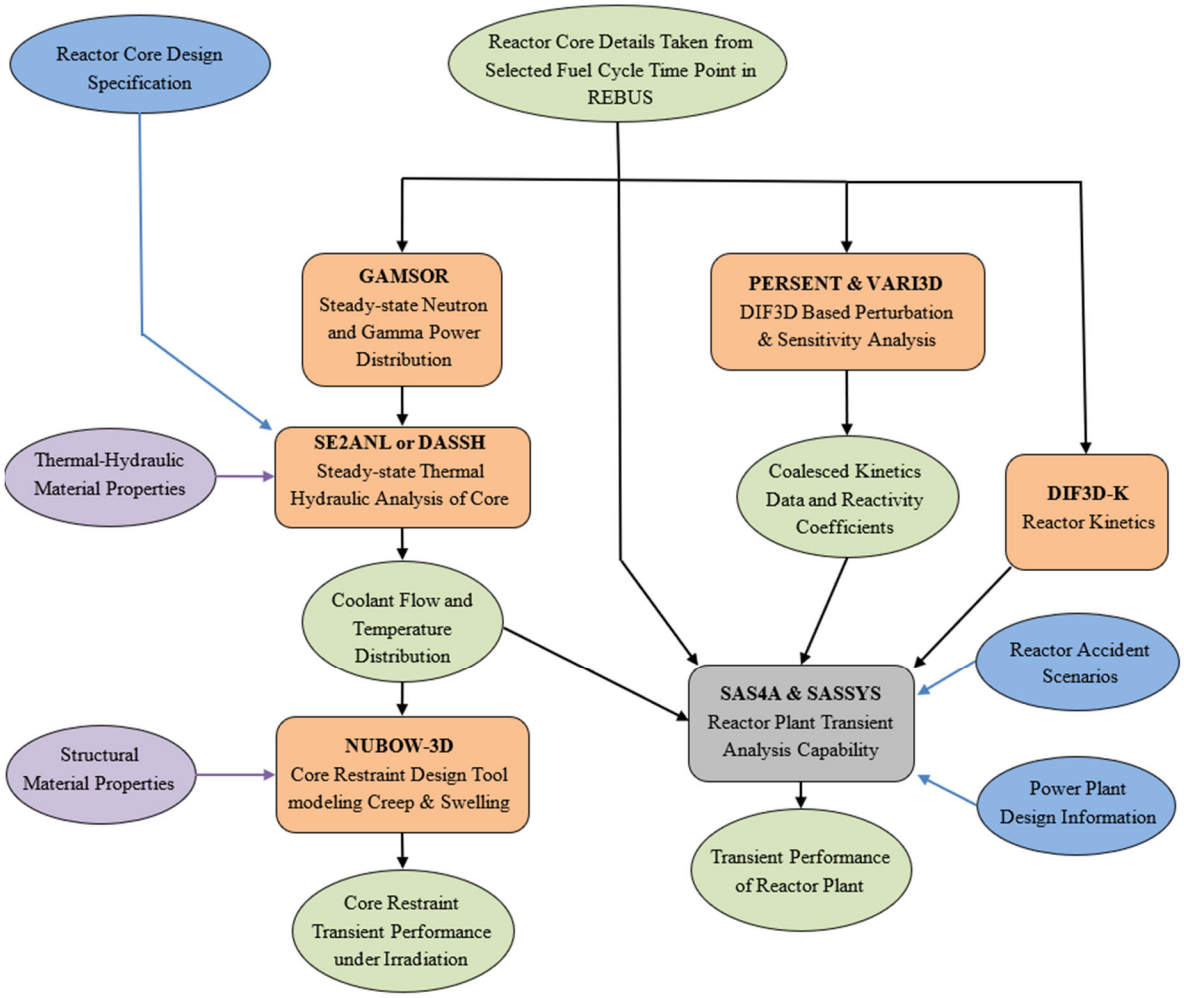

Figure 1.2: ARC Software Connections for Thermal, Mechanics, and Safety Analysis

The ETOE, MC^2-3, DIF3D, and REBUS software all have their origins in the 1960s with many of the other components added over the following 60 years of continuous development. With the exception of SUPERENERGY-2 [7], a PNNL software and common component of SE2ANL and SE2RCT, all of these programs are original creations by Argonne National Laboratory staff. A high-level summary of the capabilities of individual components of the ARC package is provided in Table 1.1 (gray indicates that the software is covered under a different SQAP).

Table 1.1: Summary Description of ARC Software Connected Components

Software Name	*Purpose*	*Software Name*	*Purpose*
MC^2-3	Generate an Equivalence Theory Based Broad Group Multigroup Cross Section Library	ETOE	Process ENDF/B data into MC^2-3 libraries
DIF3D	Steady State Neutron Particle Flux Distribution	GAMSOR	Coupled neutron-gamma Flux Solution
REBUS	Fuel Cycle Analysis	RCT	Fuel Pin Level Fuel Cycle Analysis Capability
SE2ANL	DIF3D-FD Based Steady State Thermal-hydraulics Capability	DASSH	DIF3D-VARIANT Based Steady State Core Thermal-hydraulics Capability
SE2RCT	DIF3D-Nodal Based Steady State Assembly Thermal-hydraulics Capability	NUBOW-3D	Thermal Mechanical Behavior of Irradiated System
PERSENT	DIF3D-VARIANT Based Reactivity Coefficient and Sensitivity Analysis	VARI3D	DIF3D-FD Based Reactivity Coefficient and Sensitivity Analysis
DIF3D-K	DIF3D-Nodal Based Reactor Kinetics Capability that can be Incorporated into SAS4A for Dynamics Analysis	SAS4A	Liquid Metal Coolant Reactor Safety Analysis Software

1.2 Purpose and Scope

The ARC Software Quality Assurance Program provides the controls and processes necessary to enable software improvement while meeting user and program sponsor requirements. This Software QA Plan (SQAP) delineates the SQA Program framework for the ARC software by describing the Program activities, organization, and documentation, and by clearly defining the interconnection of all program items.

It should be noted that this SQAP is aligned with the current versions of the Argonne Quality Assurance Program Plan [12] which was designed to implement the requirements of DOE O 414.1D [13].

This SQAP addresses all stages of a conventional *software life cycle*, including:

- Requirements definition,
- Software design,
- Implementation/coding,
- Qualification testing,
- Acceptance,
- Operations and sustaining engineering, and

- Software retirement.

All ARC codes are maintained by a team of software engineers, referred to herein as the *development team*. Software maintenance and operations, performed by the *development team*, is an ongoing activity. The *development team* is required to use the SQAP as a reference for Program requirements. This SQAP applies to all activities related to the *software life cycle* of the ARC codes library and associated documentation. All ARC software and documentation maintenance and modification activities must occur as per the SQAP. **The application of ARC codes to end-user needs with regard to suitability and quality is beyond the scope of this SQAP. Users are responsible for ensuring that the software is sufficient for the specified task and that the appropriate SQA measures required by their respective organizations are applied.**

The ARC *development team* is responsible for implementing *software quality assurance* requirements for the *custom-developed* software under its control. These requirements are necessary for compliance with Department of Energy (DOE) Order 414.1D, "Quality Assurance" [13], and the American Society for Mechanical Engineers (ASME) Nuclear Quality Assurance (NQA)-1-2008 with the 2009 addenda, "ASME Quality Assurance Requirements for Nuclear Facility Applications" [14, 15]. Compliance with this SQAP is required throughout the *software life cycle*, including planning, requirements, acquisition, design, implementation, *acceptance testing*, maintenance and operations, and retirement.

1.3 Assumptions and Constraints

The following assumptions and constraints are applied to all of the software discussed in this document.

- Argonne will manage the software with coordination with *acquired software* until the software is retired.
- *Software released* in accordance with this plan requires additional end-use qualification and/or dedication to be performed by the end user prior to being used in a safety or quality setting.
- All software development will adhere to Argonne policies and procedures.
- Adequate funding, required hardware, and *support software* is available to complete any planned activities.
- Roles and responsibilities cited in this plan can be reassigned as needed by division management or the Software Manager.
- All changes to the software will be controlled by the *development team*. For *release*, the Software Manager will adhere to Argonne policies and procedures.

1.4 Deviation Policy

All deviations from this plan require Software Manager approval. Whether planned or unplanned, if any deviation from this plan is necessary, the following components will be determined:

- Identification of task affected.

- Reasons for deviation defined.
- Effects on the quality of the project.
- Time and resource constraints affected.

A deviation report will be generated and authorized by the Software Manager. Deviations that violate requirements must be documented within the relevant *issue*(s).

2 References

1. C. H. Lee and W. S. Yang, “Development of Multigroup Cross Section Generation Code MC^2-3 for Fast Reactor Analysis,” Proc. of Int. Conf. on Fast Reactors and Related Fuel Cycles (FR09), Kyoto, Japan, Dec. 7-11 (2009).
2. K. L. Derstine, “DIF3D: A Code to Solve One-, Two-, and Three-Dimensional Finite-Difference Diffusion Theory Problems,” ANL-82-64, Argonne National Laboratory (1984).
3. W. S. Yang, M. A. Smith, “Theory Manual for the Fuel Cycle Analysis Code REBUS,” ANL/NE-19/21 (2020).
4. M. A. Smith, C. Adams, W. S. Yang, and E. E. Lewis, “VARI3D & PERSENT: Perturbation and Sensitivity Analysis,” ANL/NE-13/8 Rev. 4 (2022).
5. M. A. Smith, C. H. Lee, and R. N. Hill, "GAMSOR: Gamma Source Preparation and DIF3D Flux Solution," ANL/NE-16/50 revision 2 (2022).
6. W. S. Yang and M. A. Smith, "RCT: REBUS Based Pin Power Reconstruction Using the DIF3D-Nodal and DIF3D-VARIANT Options, ANL/NE-14/15 (2014).
7. K. L. Basehore, N. E. Todreas, "SUPERENERGY-2: A Multi-Assembly Steady-State Computer Code for LMFBR Core Thermal-Hydraulic Analysis," PNL-3379 (1980).
8. J. Grudzinski, T. Moran, C. Grandy, "Supplement to the NUBOW-3D Manual," ANL/NE-15/9 (2015).
9. T. A. Taiwo, “DIF3D-K: A Nodal Kinetics Code for Solving the Time-Dependent Diffusion Equation in Hexagonal-Z Geometry,” ANL/NPR-92/17, Argonne National Laboratory, October 1992.
10. Milos Atz, Micheal A. Smith, Florent Heidet, "Ducted Assembly Steady State Heat Transfer Software (DASSH) - Theory Manual", ANL/NSE-21/33, Argonne National Laboratory, (2021).
11. “The SAS4A/SASSYS-1 LMR Analysis Code System,” Argonne National Laboratory Report, ANL-FRA-1996-3 (August 1996).
12. Argonne National Laboratory, "Quality Assurance Program Plan," ANL QAPP, Current Version.
13. U.S. Department of Energy, "Quality Assurance," DOE O 414.1D, 2011.
14. American Society of Mechanical Engineers, "Quality Assurance Requirements for Nuclear Facility Applications," ASME NQA-1-2008 (2008).
15. American Society of Mechanical Engineers, "Addenda to ASME NQA-1-2008: Quality Assurance Requirements for Nuclear Facility Applications," NQA-1a-2009 (2009).

3 Definitions and Acronyms

This section provides definitions of the terms and acronyms required to understand this plan.

3.1 Definitions

Definitions commonly used in this plan are provided below.

acceptance testing: the process of exercising or evaluating a system or system component by manual or automated means to ensure that it satisfies the specified requirements and to identify differences between expected and actual results in the operating environment.

baseline: A specification or product (e.g., project plan, maintenance and operations (M&O) plan, requirements, or design) that has been formally reviewed and agreed upon, that thereafter serves as the basis for use and further development, and that can be changed only by using an approved *change control* process. [ASME NQA-1-2008 with the NQA-1a-2009 addenda edited]

change control: An element of *configuration management*, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to *configuration items* after formal establishment of their configuration identification. [ISO/IEC/IEEE 24765:2010(E)]

change request: A proposed change to a *configuration item*, including a *configuration item* stored in the *repository* or elsewhere. A *change request* that proposes a change to a *configuration item* stored in the *repository* (e.g., to report a *defect* or request an enhancement) is equivalent to a *merge request*.

configuration identification: An element of *configuration management*, consisting of selecting the *configuration items* for a system and recording their functional and physical characteristics in technical documentation.

configuration item: An item or aggregation of hardware or software (including documentation) or both that is designed to be managed as a single entity (ISO/IEC/IEEE 24765:2010(E) edited).

configuration management: A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a *configuration item*, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements (ISO/IEC/IEEE 24765:2010[E]).

corrective action: measures taken to rectify conditions adverse to quality and prevent recurrence.

defect: Anything observed in the documentation or operation that deviates from expectations based on previously verified software products or reference documents.

development documentation: Documentation of the *change control* process and *release* of *custom-developed* software.

development team: Personnel who contribute to the development of the software.

feature: A suggested improvement or enhancement to the software not associated with a *defect*.

GitLab: A web-based *revision* control hosting service for software development and code sharing.

independent reviewer: A person who reviews proposed changes to the *repository* in a *change request* and evaluates the technical adequacy of the design approach and assures internal completeness, consistency, clarity, and correctness of the software requirements and design. They must not be the individual who initiated the *change request* content.

issue: A means of reporting a *defect* or proposed enhancement of software via *GitLab*.

merge requests: Mechanism for a contributor to propose a change to a *configuration item* using *GitLab*. Note that in this context, a *merge request* is equivalent to a *change request*.

nonrecord: Material that does not meet the statutory definition of a *record* (44 U.S. Code 3301) or that has been excluded from coverage by the definition. Excluded materials are extra copies of documents kept only for reference, stocks of publications and processed documents, blank forms and library or museum materials intended solely for reference or exhibit.

qualified supplier: A supplier that has been verified by the acquiring organization as having developed the software for an intended end use under a program consistent with the American Society of Mechanical Engineers (ASME) Nuclear Quality Assurance (NQA-1) requirements.

record: Information in any form—including electronic files, created or received by an agency that falls under the legal control of the federal government—that documents an organization's functions, policies, decisions, procedures, and essential transactions; adds value to the agency; illustrates compliance with requirements; or is needed for administrative purposes or to establish quality (e.g., training qualifications).

release: A named version of software that has been developed according to this plan, has been subjected to *verification* and *validation* through a *release review*, and includes a *release log*.

release log: Document associated with a *release* that provides the *baseline* for all *configuration items* and associated reviewer information as specified in this plan.

release review: *Verification* and *validation* of all *configuration items* and *test results* for a *release*.

repository: A collection of *configuration items* that is under version control for *custom-developed* software and managed using *GitLab*.

revision. A stable snapshot of the software that has been managed according to this plan but has not undergone the process for a *release*.

software: Computer programs, associated documentation and data pertaining to the operation of a computer system, including:

- ***Acquired software*:** Software supplied through procurement, two-party agreement, or other contractual arrangements. Downloadable software that is available at no cost to the user (referred to as freeware) is also considered acquired software.

- ***Custom-developed software:*** Software built for the DOE to meet a specific set of functional requirements.
- ***Software library:*** A collection of computer program units, data, and related documentation that may be used in software development, use, or maintenance to provide functionality. These may include configuration data, help data, message templates, classes, function, subroutines and data values or type specifications.
- ***Support software:*** A computer program used in development, analysis, testing, or documenting software, including the following.
    - ***system software:*** Software designed to facilitate operation and maintenance of a computer system and its associated programs (e.g., operating systems).
    - ***software tool*:** A computer program used in development, testing, analysis, or maintenance of a program or its documentation. (e.g., compilers).

software life cycle: The activities that comprise evolution of software from conception to retirement. The software life cycle typically includes the activities associated with requirements, design, implementation, test, installation, operation, maintenance, and retirement.

software quality assurance: All actions that provide adequate confidence that software quality is achieved.

system testing: Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

test case: A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. It includes documentation specifying inputs, expected results, and a set of execution conditions for a test item.

test-driven development: A method of software development in which testing is repeatedly conducted on source code. After each test, refactoring is done and the same or a similar test is repeatedly conducted on the source code. The process is iterated until the unit functions in accordance with the specifications.

test plan: a document that describes the approach to be followed for testing a system or component.

test results: A complete set of results obtained by executing the *test cases*.

user documentation: Instructions for use describing the capabilities and intended use of the software within specified limits. May also include a theory manual, when relevant.

validation: Confirmation, through the provision of objective evidence (e.g., acceptance test), that the requirements for a specific intended use or application have been fulfilled.
[ISO/IEC/IEEE 24765:2010(E) edited]

verification: (1) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (2) Formal

proof of program correctness (e.g., requirements, design, implementation reviews, system tests). [ISO/IEC/IEEE 24765:2010(E) edited]

3.2 Acronyms

ANL	Argonne National Laboratory
ASME	American Society for Mechanical Engineers
DOE	U.S. Department of Energy
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISMS	Integrated Safety Management System
ISO	International Organization for Standardization
NQA	Nuclear Quality Assurance
QA	Quality Assurance
QL	Quality Level
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
TMS	Training Management System

4 Management

4.1 Roles and Responsibilities

Details on SQA Program roles are provided in Table 4.1. The organizational structure and interface of the SQA Program personnel with Argonne and Divisional resources is provided in Figure 4-1.Program personnel have access to the NSE Division QA Representative (QAR), who has informal oversight of the SQA Program.

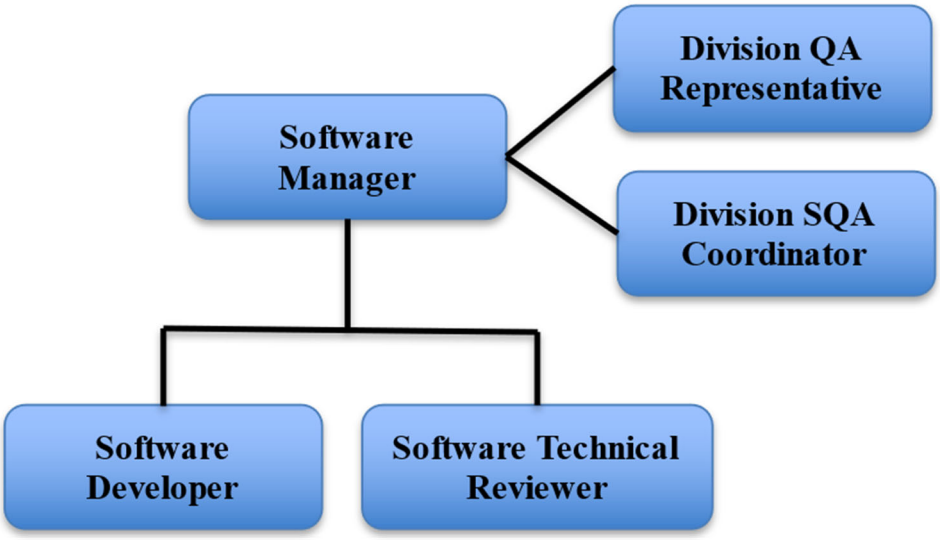

Figure 4-1 SQA Program Organization Structure

Table 4.1: Roles and Responsibilities

Role	Tasks and Responsibilities
Software Manager	• Reassign role and responsibility in this plan, as needed. • Schedule work activities as needed. • Assign resources as needed. • Manage budget as needed. • Establish schedule for *change request* as needed. • Identify members of the software *development team* who will serve as software developers and technical reviewers for development and *defect* related *change request* work. • Manage schedule and scope. • Coordinate and manage software *releases*. • Establish timeline for completion of *change request*, as needed. • Assess adequacy of *change control* process, as needed. • Document and control technical requirements/design per this plan. • Acquire IT materials and services in accordance with ESHQ-QA-

	7.3 and the AMOS workflow; for software acquisitions, ensure that procurement documents identify the mechanism for supplier reporting of software errors to the purchaser, and the purchases reporting of software errors to the supplier. • Document and control test procedures and instructions for user per this plan. • Manage and resolve problems per this plan. • Build, configure, and test IT assets per this plan. • Place software under version control. • Perform acceptance test, document review and approval of *test results* in accordance with this plan. • Disposition and maintain all *record*s per this plan. • Ensure the implementation of the required SQA training for members of the software *development team*. • Document and maintain this plan (SQAP). • Authenticate *record*s.
Software Development Team Members	• Manage *configuration items* within the *repository* and software libraries. • Manage modifications to the software. • Oversee problem reporting. • Act as *independent reviewer*. • Coordinate with an *independent reviewer* to perform and document a review that evaluates the technical adequacy of the design approach and assures internal completeness, consistency, clarity, and correctness of the software requirement/design. • Approve the technical requirements/design. • Manage technical requirements/design in accordance with this plan • Close *change requests*. • Document *change requests* including change descriptions. • Approve or disapprove *change request* and inform requestor. • If change impacts technical requirements/design, approve revised technical requirements/design.

4.2 External Interfaces

In order to support collaboration, two external interfaces to the SQA program are in place which include external software developers and external software users.

For external users and developers, access to the *repository* and the digital resources are limited to those that have been granted privileged access. As defined in this plan, all *issue*s and *change requests* must go through an independent review by a member of the software *development team*. Therefore, an individual with the role of external software developer is

not required to undergo the training defined in Section 13 of this plan. It is the responsibility of the software manager to ensure that the correct procedure is followed for software development and that all aspects of an *issue* and/or *change request* are compliant with this plan.

5 Documentation

Two classes of documentation are defined: standard documentation and *development documentation*. At a minimum, the documentation listed in the following sections are required. From this point forward, the term "documentation" alone shall refer to standard documentation only.

5.1 Standard Documentation

Standard documentation is managed and maintained within the *repository* in accordance with Section 12. Where appropriate, documentation is generated using Argonne templates.

- Software Quality Assurance Plan (SQAP; this plan), which includes the *configuration management* plan, software *verification* and *validation* plan
- *User documentation* (e.g. user and/or theory manual)
- Software requirement specification
- Software design description
- Software *test plan*
- Failure analysis report
- Communication and contact information
- Software coding standards
- Software Library List

5.2 Development Documentation

The following documents shall be maintained to document the *change control* process leading to a *release*. Development documentation is managed as *record*s in accordance with Section 12.

- *Change Request(s)*
- *Release log(s)*
- *Issues*

6 Reviews

At a minimum, the following reviews shall be conducted.

6.1 Documentation Reviews

Documentation reviews are performed following processes dictated by the *configuration management* system as detailed in Section 7.2.

6.2 Software Reviews

6.2.1 Design Verification Review

A review shall be conducted by an *independent reviewer*. The reviewer may rely on assistance from any individual, but the final approval or disapproval is the sole responsibility of the *independent reviewer*. All review comments are retained in the *GitLab* system.

All reviews and approvals will be recorded within the *change request* using the integrated approval system of *GitLab*. This includes identification of the *independent reviewer* prior to acceptance of the proposed modification. Any proposed changes to the *repository* trigger automated testing. A summary of the automated testing results is retained in the *GitLab* system.

An *independent reviewer* shall conduct a design review for all *change requests* and shall evaluate the technical adequacy of the design approach and ensure internal completeness, consistency, clarity, and correctness of the software design and demonstrate that software design is traceable to the software requirements. This review will be conducted as specified in Section 9.4.1. Review of *test results* are considered to be within the scope of the design *verification* review.

6.3 Release Review

A *release* requires an additional *release review*, as specified in Section 9.6.2, to ensure compliance with the approved software requirements. Review of *test results* are considered to be within the scope of the *release review*.

7 Configuration Management

Software *configuration management* activities, including *configuration identification*, *change control* , status accounting, and software configuration reviews, are established during the planning phase of the *software life cycle* and implemented through operations and maintenance until the software is retired.

7.1 Identifying Configuration Items

The *configuration items* include:

- Documentation as defined in Section 5.1
- The *configuration items* within the *repository* include the code necessary to satisfy the software requirements as well as the *repository* documentation. The identification of *configuration items* within the *repository* is inherent to the *change control* process and is the responsibility of the *independent reviewer*.
- *Support software* and Software Libraries as defined in Section 11.

The software *baseline* is the latest *release* unless otherwise specified within a *change request*.

7.2 Managing Configuration Items

Documentation shall be managed by the software manager as follows:

- The SQAP (this plan) shall be reviewed at a minimum of every three years. Modifications to the SQAP (this plan) require an approval from the software *development team* and the relevant QA representatives. This plan is retained as a *record* in the SQA *repository*.

The *configuration item* within the *repository* will be managed by the software *development team* and adhere to the *change control* process detailed in Section 9.8.2 and maintained under *configuration management* until the software is retired.

Software libraries will be managed by the software *development team* and adhere to the *change control* process detailed in Section 9.8.2. Any external software libraries required shall be listed in the Software Library List.

7.3 Naming Configuration Items

Any new documentation shall follow the naming convention “XYZ-Title-Revision Number”, where XYZ is the associated software (e.g. DIF3D, REBUS, etc...), “Title” is a descriptive title of the document, and N is the *revision* number. Documentation will be recorded in the *release log* during the *release* process as detailed in Section 9.6.

The *configuration items* included in the *repository* are uniquely identifiable for all *revisions* or *releases* by the inherent capabilities of the *GitLab configuration management* system.

Support software and external software libraries will be named and versioned as defined by the supplier of the software and will be recorded in the *release log* during the *release* process as detailed in Section 9.6.

All *releases* shall be identified by a tag based on the date of *release*. The format shall be "YYYY-MM-DD", where YYYY is the four-digit year, MM is the two-digit month, and DD is the two-digit day of the month. If a *release* is altered via a correction action, that tag for the patched version shall include a patch number as YYYY-MM-DD-pN, where "N" is the patch number.

7.4 Software Change Control

Changes to the software *baseline* will be managed in accordance with Section 9.8.2.

7.5 Software Configuration Status Accounting

Configuration status accounting activities record and report the status of a *change request*. A *change request* remains in proposed status until it has been approved by an *independent reviewer* or closed. All *change requests* are tracked for the full duration of their lifetimes, irrespective of status, using the *GitLab* system.

A *change request* can have one of the following statuses:

- Proposed: A *change request* that is under development by a contributor and reviewed by an *independent reviewer*. Within *GitLab*, all *change requests* that are "open" have this status.
- Approved: A *change request* that has been reviewed as detailed in Section 9.4.1 and approved by an *independent reviewer* using *GitLab* approval system. The changes may be merged by any member of the software *development team* after approval.
- Completed: An approved *change request* that has been merged.
- Disapproved: A *change request* that no longer has a proposed status and has not been merged. Within *GitLab* all *change requests* that are "closed" and have not been merged have this status.

7.6 Software Configuration Audits

Surveillance of the *repository* is completed regularly to verify compliance with this Configuration Management Plan.

For each *release*, functional configuration audits are achieved through the performance of the required reviews and approvals specified in this plan. Physical configuration audits are the responsibility of the end user.

8 Software Acquisition

Software or software services acquired to support the software will be procured with support from the ANL procurement office in accordance with LMS-MNL-23. *Acquired software* will be managed upon receipt, and, as such, will be considered a *configuration item* as defined in Section 7.1.

Procurement documents associated with acquisitions from a *qualified supplier* shall identify requirements for supplier's reporting of software errors to the team and, as appropriate, the teams' reporting of software errors to the supplier. For software acquisitions where there is no relationship with the supplier, the project lead shall determine the correct approach for obtaining error information from a supplier (e.g., periodic monitoring of a supplier's website).

Maintenance and support agreements will be put in place with suppliers, as applicable, for software updates and *revisions* available from a supplier.

When acquiring software (including upgrades and software libraries), the following will be incorporated into the life-cycle documentation:

- Requirements describing capabilities, limitations, and intended use;
- A *test plan* and corresponding *test cases* that will be used to verify requirements of the *acquired software* are met; and
- Instructions for use of the *acquired software*.

The resulting documentation and associated software will become part of the current *baseline* and managed in accordance with the *change control* process detailed in Section 9.8.2.

Any *software library* required shall be listed in the Software Library List.

9 Software Engineering Method

All development and maintenance of the software uses *test-driven development*. The testing process ensures all contributions are fully tested before being integrated into the main branch of the *repository*. Only a *revision*, or series of *revision*s, that follows the *release* process (see Section 9.6) will become a new *baseline* for the software.

All *revision*s and *release*s are stored using *GitLab* and are available for download from this location at any time.

The tasks delineated in the following subsections encompass SQA activities performed throughout the *software life cycle*.

9.1 Project Initiation

9.1.1 Work Activities and Schedule Allocation

Work activities will be coordinated with the software *development team* and formally defined by the software manager as a general work activity/schedule with high-level milestones for the fiscal year. Detailed planning will be conducted at a frequency deemed appropriate by the software manager.

9.1.2 Resource Allocation

The software manager will ensure adequate resources are available to the software *development team* for the work described in Section 4.

9.1.3 Budget Allocation

The development budget is based on year-to-year customer needs and funding granted by research and development activities. The current year budget is managed and can be obtained by contacting the software manager.

9.2 Software Requirements

The requirements for the software should align with the work activities (see Section 9.1.1). The requirements shall be recorded within the *repository*, defined within *test cases*, and available in a *software requirements specification* document.

The software requirements shall identify operating systems, functions, interfaces, performance requirements, installation considerations, design inputs, and any design constraints of the computer program.

9.2.1 Requirements Traceability

The satisfaction of the software requirements will be documented in the software *verification* and *validation* reports. Each *test case* identified in the software *verification* report should specifically reference the requirement that is being tested. With this, the set of *test cases* ensures the software

continuously meets the requirements laid out for the software. The complete set of *test results* shall be accessible through the software *verification* and *validation* reports.

9.3 Software Design

The software design shall consider the software's operating environment. Measures to mitigate the consequences of problems, as identified through analysis, shall be an integral part of the design. These potential problems include external and internal abnormal conditions and events that can affect the software. The software design shall define the computational sequence necessary to meet the software requirements and include, as applicable, numerical methods, mathematical models, physical models, control flow, control logic, data flow, process flow, data structures, process structures, and the applicable relationships between data structures and process structures.

Design documents shall be stored within the *repository*.

9.4 Software Implementation

Contributors utilize the software design to implement the relevant *feature*s for the software applications. The code standards referenced in Section 10.1 must be followed for all code developed or modified as part of a *change request*.

The software *development team* can accept contributed code from members outside of Argonne provided that the code meets all coding standards, is independently reviewed, and provides adequate *test cases* that include adequate design and requirements.

9.4.1 Independent Review

Independence is achieved through a variety of measures that are integrated in the development and *change control* process. Each *change request* must be peer reviewed by an *independent reviewer*. The *test results* act as an independent indicator of the software performance from the perspectives of expected results, expected errors, and other factors deemed necessary by the reviewer.

This review includes evaluation of the *test results* as detailed in Section 9.5.4. The review shall ensure that requirements, design, and implementation phase activities have been satisfactorily completed and the software is approved to follow the *release* process. However, a *release* will only be performed if deemed necessary by the software manager. The review documentation is maintained on *GitLab*.

9.5 Software Verification, Validation, and Testing

Software *verification*, *validation*, and testing activities are performed to ensure compliance with software engineering requirements and adequacy of the software design. All *verification*, *validation*, and testing activities are expected to be performed by the automated testing apparatus and any additional testing done by the *independent reviewer* should be documented as part of the review process.

9.5.1 Software Test Plan

9.5.1.1 Test Cases

As part of a *change request*, contributors are expected to identify or add at least one *test case* that covers the implemented code changes. An *independent reviewer* must approve a *change request* and ensure that the *change request* includes documentation of the software changes, documentation on any changes to the software requirements, and adequate testing of the software. The process for handling *change requests* that modify requirements is defined in Section 9.8.2.

Tests cases specify what a test should do, the inputs, and the post-conditions for determining test success or failure and assuring that the software produces expected results. *Test cases* shall ensure that the software application properly handles abnormal conditions and credible software application failures. Additionally, *test cases* shall ensure the software application does not perform adverse untended functions nor degrade other software application running in the operating environment.

Each *test case* will:

- Define the requirement that the test satisfies
- Reference one or more design documents
- Reference at least one of the following:
    - An *issue* that motivated the associated *test case*.
    - The *change request* that implemented the *test case*.

Acceptance criteria for each test are defined in the *test case* definition.

9.5.2 Software Test Execution

Testing is performed automatically as part of the *change control* process when a contributor creates a *change request*. Table 9.1 summarizes the various levels of testing that occur during the *change control* process.

Automated testing will indicate successful execution of existing *test cases*. This ensures that the software demonstrates adherence to the documented requirements and that the software produces correct results. A summary of the automated testing results is viewable in *GitLab*.

Table 9.1 Summary of testing and reviews that occur during the *change control* process.

Step	Description	Who	What
1	*Change Request* Testing	Automated	This testing is executed when a contributor submits a *change request* through *GitLab*. The proposed change is checked for adherence to coding standards, compiled, and tested. The *test results* are viewable for the *change request* within *GitLab*. See Section 9.8.2.3 for details.
2	Code Review	Independent reviewer	All code modifications to the *repository* go through a review by an *independent reviewer*, as detailed in Section 9.4.1. After approval the *change request* is merged into the development branch of the *repository*.
3	Development Branch Testing	Automated	This step re-evaluates *test cases* to ensure that merged *change requests* are compatible. All tests identified in the *verification* documentation for the software should be executed at this time. The details of the testing will be maintained as a *record*. If all tests pass, the changes are automatically merged to the stable branch. If not, a new *change request* must be created to investigate and correct any identified problems. The merge to the stable branch results in a *revision*.
4	Documentation	Developer	The documentation for the new *revision* is made available.
5	*Release*	Software Manager	As necessary, the *release* process will be followed to establish a new *baseline* as described in Section 9.6.1.

9.5.3 Test Results

Test results are accessible within *GitLab* and are readily available to the *independent reviewer* during the review process discussed in Section 9.4.1. Each *test case* defines the information that shall be recorded in a test result. *Test results* for each *release* are managed as defined in Section 9.6.

9.5.4 Test Results Evaluation

Test result evaluation occurs during the evaluation of a *change request* and is performed by an *independent reviewer*. The *independent reviewer* shall be responsible for verifying that the *test cases* are sufficient for the *change request* such that each *test case* fulfills at least one requirement and all

requirements have at least one *test case*. If all *test cases* pass then all of the software requirements have been satisfied.

9.6 Product Release

Software intended for use in a safety or quality setting must undergo the *release* process detailed in this section. The software *release* is supported for at least the duration between one *release* and the next.

*Revision*s present in the *repository*, as detailed in Table 9.1, are not intended to satisfy any quality standards. The *revision* is provided "as-is," and no guarantee of functionality or performance of a *revision* is provided.

9.6.1 Release Candidate Identification and Control

Any *revision* can be selected as a candidate for *release*.

9.6.2 Release Review

The candidate *revision* undergoes a *release* review as detailed in this section. The software manager, who shall be familiar with the design, verifies that all *test cases* have passed under all relevant configurations defined in the software *test plan*, and reviews each of the documents identified in Section 5 to verify that they are accurate and complete.

The software manager shall create a *release log* that includes what is relevant and important for acceptability of the software product. The following information, which is not included in the *test results* is included with the *release* during approval.

- Date of the *test results*.
- Person evaluating the *test results*.
- Evidence that any unexpected or unintended *test results* have been dispositioned.
- Any actions taken in connection with any deviations from this plan.
- A list of all *configuration items*, as defined in Section 7.1, with the following exception: (1) *configuration item*s in the *repository* and (2) software libraries. The ability to list the *configuration item*s in the exceptions is inherent to the version control system
    - A list of the platforms/version and all the *support software* and libraries will be made easily accessible on the main page.
- Evidence that all software quality documentation listed in Section 5 has been reviewed for completeness and consistency.
- Acceptability.

9.6.3 Release Approval

Following the successful completion of testing, the software manager approves the *release* by:

- Creating a new labeled branch from the *revision*. The creation of this branch prohibits further development or modification of the *release*.

- Adding the *release log* to the labeled branch.
- Adding the associated *test results* to the labeled branch.
- The branch is tagged with the unique identifier for the *release* in accordance with Section 7.3

Completion of these operations establishes a *release* and represents the approval of the *release*. Each *release* is added to the controlled *baseline*. Based on the inherent capabilities of the *repository* and the *release log*, the *configuration item*s for a specific *release* are accessible given a unique tag.

Upon *release* approval, a copy of the *repository* documentation for the *release* may be made available. Note, this is a copy of the content that exists in the *release* and is added for convenience.

*Release*s may receive patches to resolve *defect*s but may not receive new enhancements. Patches are managed in accordance with Section 9.8.2 and named as detailed in Section 7.3.

9.7 Product Acceptance

It is incumbent upon any external organization using this software to conduct final *acceptance testing* for their specific end-use prior to implementing any *release* in their software environment.

9.8 Operations and Maintenance

After *release*, the software shall be controlled in accordance with the user organization's approved procedures and instructions.

9.8.1 Problem Reporting and Corrective Action

Any individual can report a problem. A problem should be reported in one of three ways as detailed in the communication and contact information list (see Section 9.8.3):

- Direct communication
- Creation of an *issue*
- Creation of a *merge request*

A problem is evaluated by a member of the software *development team* as either a *defect*, an enhancement, or a problem that was reported by mistake (a user mistake). The categorization is reported to the user as detailed in Section 9.8.3.

If a problem is determined to be a *defect*, an *issue* will be created by a member of the software *development team* (if it was not reported via that mechanism). The *change control* process defined in Section 9.8.2 will be followed.

If a problem is determined to be an enhancement, and is approved as a desirable part of the software design by the software manager, an *issue* will be created by a member of the software *development team* (if it was not reported via that mechanism). The *change control* process defined in Section 9.8.2 will be followed.

Communication regarding a *defect* and *corrective action* shall be performed as detailed in Section 9.8.3.

The *corrective action*s within the scope are tracked using the *change control* process, as defined in Section 9.8.2. Defect resolution may be tracked via the original *change request* related to the modification that led to the *defect*.

9.8.2 Software Change Control

All *configuration item*s defined in Section 7.1 are under *change control*. The *change control* process varies based on the *configuration item*, as defined in Section 7.2. This section details the process for a *change request*, also known as a *merge request*, which is the *change control* process for support libraries and *configuration items* within the *repository*.

A change can be initiated because of a *defect* that must be corrected, or an enhancement due to:

- External or regulatory changes that result in new software requirements
- Internal changes that result in new software requirements or design
- Upgrades for performance, adaptability, etc.
- New technologies that need to be incorporated
- Software refactoring
- Vulnerability patches
- Changes in the operating environment

Ideally, external contributors will communicate with the software manager to work on existing approved *issue*s that are viewable on *GitLab*. However, the process is defined to support development efforts that begin outside of a traditional *change control* process. This presents a risk that an external contributor will propose changes that include complete code implementations and are rejected after they have expended significant effort. Instructions for external developers to contribute software changes shall be included in the communication and contact information list. Contributors are encouraged to reach out to the software manager prior to implementation.

9.8.2.1 Initiating Changes

The process for initiating a change is flexible.

Initiation occurs in one of two ways. Note that either of the following two initiation mechanisms requires a template to be completed with the required information for an enhancement or a *defect* (see “Enhancements” and “Defects” below):

- Opening an *issue*.
- Creating a *merge request*.

All *issue*s and *merge request*s shall be evaluated by a member of the software *development team* and determined to be a *defect*, an enhancement, or closed if the initiation was created as result of a user mistake. Both shall include:

- Unique identifying number generated by *GitLab*.
- The identity of the creator and the creation date.
- Optional comment thread is available to provide feedback and engage in discourse with the *issue* submitter.

A member of the software *development team* shall ensure that the required information is added to the *issue* or *merge request*. Templates for *issue*s and *merge request*s will be present in the *GitLab repository* to help ensure this required information is provided.

Enhancements

An enhancement shall include the following and shall be recorded in an *issue* or *merge request*:

- The reason for the change.
- A concise description of the desired change.
- The impact of the change on existing *configuration item*s.

Defects

All *defect*s are captured by creating an *issue* if it was not previously created, either by the user or a member of the software *development team*. Each *issue* shall capture the following information regarding a *defect*:

- A concise description of the *defect*.
- The steps necessary to reproduce the *defect*.
- The impact of the change on existing *configuration item*s.

A member of the software *development team* will evaluate the impact of a *defect*. If possible, other unique and/or significant information about the *defect* that will aid in the evaluation will be included: for example, limitations and capability difference between versions or anticipated new versions. The *issue* shall be tagged as a "bug" within *GitLab*.

A member of the software *development team* will determine the level of the *defect* as defined below.

- Critical. An *issue* that affects the accuracy of the results, past and/or present, and requires immediate attention.
- Normal. Issues affecting the operation or execution of the code, with a low possibility of significantly affecting the results.
- Minor. Issues that do not affect the accuracy of the results.

If the *defect* is critical, users will be notified and provided the relevant information, including the impact of the *defect*, information on how to avoid the *defect*, *corrective action*s, and when *corrective action*s will be implemented.

The *defect* resolution will be contained within a *merge request* and be associated with the *issue*.

9.8.2.2 Implementing Changes

The *merge request* process is followed for all *change requests*. This process includes requirements, design, implementation, testing, and independent review.

Any contributor can choose to work on a particular *issue* by associating the *issue* with a *change request* in *GitLab*. A listing of all *issue*s is available through *GitLab*.

If the *change request* requires software changes, the *independent reviewer* will ensure appropriate *test cases*, requirements, design documentation, and appropriate cross-referencing are all included within the *change request*. The tests performed by the automated testing as defined in the software *test plan* enforce the inclusion of each of these components.

The software manager may require a schedule to be established for a *change request* that includes planning for project funding, labor, and evaluation of impacts to work activities.

9.8.2.3 Cursory Evaluation

Once the *change request* is initiated, automated testing begins the process of running checks on the proposed changes. Namely, to the extent possible, the coding standards are enforced, including proper *issue* refencing, spacing, size checks, etc. as defined in the software *test plan*. These are performed to provide prompt feedback to the contributor. This process simplifies the job of the *independent reviewer* because evaluation of changes is generally deferred until testing is completed.

Table 9.1 provides a depiction of the testing and reviews associated with the *change control* process for a *change request*. It is important to note that every approved *change request* does not result in a *release*. The *change request* process in Steps 1-4 in Table 9.1will occur many times prior to a *release* (Step 5).

9.8.2.4 Evaluating Changes

Deferring the evaluation until testing is complete provides assurance to the reviewer that impacts are indeed within the expected scope for a particular *change request*.

The *independent reviewer* controls and is responsible for the evaluation and disposition of proposed changes for all *configuration item*s within the *repository*. The *independent reviewer* will consider the impact of the proposed change and assign actions appropriate to the level of impact. If the proposed change is disapproved, the decision will be noted on the *change request*. If additional information is needed, it will be noted and returned to the contributor for completion. The contributor is then responsible for editing the content as requested by the *independent reviewer*.

It is the responsibility of the contributor to monitor the *change request* for communications.

There is no established time for review of the *change request*s; however, consideration should be given to the priority established by the software *development team*.

9.8.2.5 Approving or Disapproving Changes

After a *change request* is reviewed, the *independent reviewer* will determine if the proposed change will be approved or disapproved, or if additional work will be requested. The decision (following the

definitions in Section 7.5), the name of the software *development team* member giving the final disposition, and the date of the disposition shall be recorded (this is automatic within *GitLab*).

If approved, the proposed changes associated with the *change request* are merged into the development branch. The resulting modification will follow the process shown in Table 9.1 and become part of a *revision*. The change will be included in the next *release*, as detailed in Section 9.6.

9.8.3 Communication

Methods of communication with contributors and users shall be specified in a communication and contact information list.

The communication and contact information list shall include:

- Instruction for reporting a problem and monitoring *corrective action*s (see Section 9.8.1)
- Instruction for contributing to the software, see Section 9.8.2.

9.9 Software Retirement

Users shall be responsible for the retirement of software for their intended use using their internal procedures.

The software manager shall determine when software support shall be discontinued. At that time, a retirement plan shall be documented and approved to describe how the following activities will be completed, as applicable:

- This plan and any associated controlled documents will be updated to reflect the change in disposition. If all software within the scope of this plan is retired, this plan and all associated controlled documents will be managed in accordance with the *record*s disposition schedule.
- Access to the software will be terminated.
- The retired software will be archived within the *GitLab* infrastructure.
- All affected websites and links to the retired software shall be updated to reflect the status of the software.
- Support will be discontinued, and notifications sent to all affected users.

10 Standards, Practices, Conventions, and Metrics

10.1 Software Coding Standards

Any new software changes and developments shall follow the same software coding standard as the original source code. The software technical reviewer shall ensure that the applicable software coding standard has been followed before the proposed change(s) are merged.

10.2 Methods, Techniques, and Tools

It is necessary to use modern software-development tools, methodologies, and techniques to maintain the usefulness of the software to the user community. Methods, techniques, and tools are identified below.

A *test-driven development* process is used through *merge request*s. Within this process, various techniques – including code coverage analysis; *acceptance testing*; peer reviews; and automated builds – enable contributors to perform rapid, incremental changes to meet project milestones. The process is enabled by modern tools such as the Git version control system, *GitLab repository* services, and a range of Python-based tools for testing and documentation.

11 Support Software

The use of *support software* shall be limited to the items identified in this section.

11.1 GitLab Repository

The *GitLab* service provides online distributed development using Git-based repositories. Services include issue tracking, testing integration, push notifications for events, permissions, and backup.

11.2 System Software

System software is identified by the software manager and documented within the results associated with a *release*, as detailed in Section 9.6.2.

Any changes to associated *system software* will be tested with the automated testing capability.

12 Records Collection, Maintenance, and Retention

The standard documentation and development quality *record*s identified in Section 5 will be held within protected branches in *GitLab* repositories.

The git repositories housed on the git-nse server located within the enterprise data center in building 338. The git-nse server is incrementally backed up daily using an IBM Tivoli/Storeserver Backup appliance, or an equivalent tool. Backup are exported to tapes weekly and stored within a vault in a separate building.

The quality *record*s shall be maintained for 25 years after the retirement of the software in accordance with the schedule outlined in DOE R&D Records Schedule N1-434-08-2, N1-434-96-9, and N1-434-07-01.

12.1 Records Authentication

Quality *record*s are authenticated by the software manager by verifying that all documents identified in Section 5 are accurate and complete in accordance with Section 9.6.3. For *record*s retained in *GitLab*, the *change request* process described in Section 9.8.2 includes the approval cycle and the control points where approval signatures are captured.

13 Training

The software manager is responsible to ensure that all members of the software *development team* are qualified and trained in accordance with LMS-PROC-16 “Mandatory Training,” and that their training is recorded in Argonne’s Training Management System (TMS). Each role identified in Table 4.1 in this plan shall complete the EQO203 “Quality Assurance Program Plan General Training” course.

Software *development team* members must complete training prior to beginning work. Other roles may conduct work prior to all training being completed under supervision of an individual with equivalent training.

A list of personnel who are assigned a specific role will be maintained and controlled by the software manager. This list shall be reviewed and updated with at least each *release*, or as deemed necessary by the software manager.

Nuclear Science & Engineering Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 208
Argonne, IL 60439-4842

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC