DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. Reference herein to any social initiative (including but not limited to Diversity, Equity, and Inclusion (DEI); Community Benefits Plans (CBP); Justice 40; etc.) is made by the Author independent of any current requirement by the United States Government and does not constitute or imply endorsement, recommendation, or support by the United States Government or any agency thereof.

ANL/NSE-25/65

Argonne NATIONAL LABORATORY

Software Quality Assurance Plan

Cardinal

Nuclear Science and Engineering Division

ANL/NSE-25/65

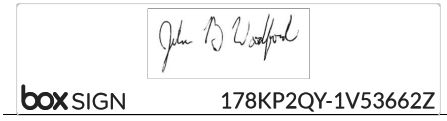
Software Quality Assurance Plan

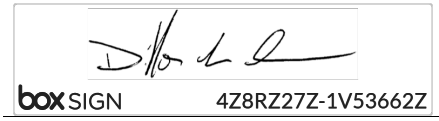
Cardinal

by
Kalin R. Kiesling
Nuclear Science and Engineering Division, Argonne National Laboratory

OCTOBER 2025

Prepared by:  Date: Nov 5, 2025
Kalin Kiesling
Cardinal Software Quality Assurance Lead

Reviewed by:  Date: Nov 7, 2025
John Woodford
NSE QAR

Approved by:  Date: Nov 7, 2025
Dillon Shaver
Cardinal Software Co-Manager

Approved by:  Date: Nov 8, 2025
April Novak
Cardinal Software Co-Manager

TABLE OF CONTENTS

LIST OF TABLES

1 Introduction

1.1 Project Background

The software Cardinal is an open source, advanced, and modern multiphysics analysis tool with high-resolution thermal-hydraulics and radiation transport capabilities [1]. Cardinal is under development at Argonne National Laboratory under the U.S. DOE Office of Nuclear Energy's Nuclear Energy Advanced Modeling and Simulation (NEAMS) program. The software is a wrapping of the GPU-oriented spectral element Computational Fluid Dynamics (CFD) code NekRS [2] and the Monte Carlo particle transport code OpenMC [3] within the object-oriented application framework MOOSE [4]. Multiphysics feedback is implemented in a geometry-agnostic manner which eliminates the need for rigid one-to-one mappings. A generic data transfer implementation also allows NekRS and OpenMC to couple to *any* MOOSE application, enabling a broad set of multiphysics capabilities. Simulations can also leverage combinations of MPI, OpenMP, and GPU resources.

Cardinal development aims for advances in physical modeling, numerical methods, and software engineering to enhance its user experience and usability for reactor analyses. Cardinal utilizes an object-oriented application framework (MOOSE), and its underlying meshing and finite-element library (libMesh [5]) and linear and non-linear solvers (PETSc [6]) to leverage modern advanced software environments and numerical methods. This approach facilitates code development as well as capability expansion and standardization.

Cardinal supports general, physics-agnostic data transfers between NekRS, OpenMC, and MOOSE to enable a wide variety of multiscale and multiphysics simulations. Supported physics capabilities for NekRS include conjugate heat transfer, volumetric power/temperature/density feedback, and 1-D/3-D coupling to systems codes. Cardinal also interfaces NekRS to MOOSE's stochastic tools module for forward uncertainty quantification. For OpenMC, Cardinal supports both cell- and mesh-based tallies coupled to MOOSE for Constructive Solid Geometry (CSG) and DAGMC Computer Aided Design (CAD) geometry [7]. In addition, mesh-based geometries in OpenMC support on-the-fly deformation using displacements computed by MOOSE's solid mechanics module for considering reactivity feedback from thermal expansion/bowing. OpenMC's tallies are also integrated with libMesh's Adaptive Mesh Refinement (AMR) system for on-the-fly refinement and coarsening. And by interfacing NekRS and OpenMC with MOOSE's postprocessing, stochastic simulation, and other modules, users have a wide range of data analysis, multiscale closure development, and engineering workflow capabilities at their disposal.

The key features of Cardinal include:

- Robust and high-order spatial and temporal discretization methods for fluid flow, heat transfer, and radiation transport modelling.
- Multi-scale modelling and flexible coupling between multiple physics, enabling a wide range of engineering analyses and convenient integration with other advanced or conventional simulation tools.
- Engineering postprocessing utilities for generating multiscale closure data and simulation analysis.

1.2 Purpose and Scope

The Cardinal Software Quality Assurance (SQA) Program aims to provide the controls and processes necessary to enable continuous, high-quality software development while meeting user and program sponsor requirements. This SQA Plan (SQAP) delineates the SQA Program framework for Cardinal 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 version of the Argonne Quality Assurance Program Plan [8], which was designed to align with DOE O 414.1D [9]. This SQAP is also aligned with the revision 10 of the SQAP for MOOSE and MOOSE-based applications [10].

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.

Cardinal is maintained by a team of software engineers, referred to herein as the development team (see definition). Cardinal 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 and guidance on the appropriate use of procedures. This SQAP applies to all activities related to the software life cycle of the Cardinal code library and associated documentation. All Cardinal software and documentation maintenance and modification activities must occur as per the SQAP. **The application of Cardinal to end-user needs regarding 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 Cardinal development team is responsible for implementing software quality assurance (SQA, see definition) requirements for software (see definition) under its control. These requirements are necessary for compliance with Department of Energy (DOE) Order 414.1D, "Quality Assurance" [9], and the American Society for Mechanical Engineers (ASME) Nuclear Quality Assurance (NQA)-1-2022, "Quality Assurance Requirements for Nuclear Facility Applications" [11]. Compliance with this SQAP is required throughout the software life cycle (see definition), including planning, requirements, acquisition, design, implementation, acceptance testing, maintenance and operations, and retirement.

1.3 Assumptions and Constraints

- Argonne will manage the software with support from collaborators until the software is retired.
- Per Argonne's Quality Assurance Program Plan [8] and Quality Manual [12], Cardinal is designated as a Quality Level D (QL-D) non-safety software and is managed accordingly.

- Cardinal development will adhere to ANL's SCITECH-8, "Material Published Externally" [13], where applicable.
- Adequate funding, required hardware, and support software (see definition) is available to complete planned Cardinal activities.
- Roles and responsibilities cited in this plan can be reassigned as needed by the software managers.
- All changes to Cardinal, including security patches, will be controlled through the Change Control Board (CCB, see definition). For release, the project lead will adhere to SCITECH-8 [13].
- The scope of this document covers the Cardinal software and its dependency on MOOSE [7], OpenMC [3] and NekRS [2].

1.4 Deviation Policy

All deviations from this plan require software manager(s) 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 software manager will document, either through GitHub or other retained methods, of the deviation and their approval for the deviation. Deviations that do not fulfill requirements must be documented within the relevant issue(s) (see definition).

2 References

[1] A. J. Novak *et al.*, “Coupled Monte Carlo and Thermal-Fluid Modeling of High Temperature Gas Reactors Using Cardinal,” *Ann. Nucl. Energy*, vol. 177, p. 109310, 2022, doi: 10.1016/j.anucene.2022.109310.
[2] P. Fischer *et al.*, “NekRS, a GPU-accelerated spectral element Navier–Stokes solver,” *Parallel Comput.*, vol. 114, p. 102982, 2022, doi: https://doi.org/10.1016/j.parco.2022.102982.
[3] P. K. Romano, N. E, B. R. Herman, A. G. Nelson, B. Forget, and K. Smith, “OpenMC: A state-of-the-art Monte Carlo code for research and development,” *Ann. Nucl. Energy*, vol. 82, pp. 90–97, Aug. 2015, doi: 10.1016/j.anucene.2014.07.048.
[4] L. Harbour *et al.*, “4.0 MOOSE: Enabling massively parallel Multiphysics simulation,” *SoftwareX*, vol. 31, p. 102264, 2025, doi: https://doi.org/10.1016/j.softx.2025.102264.
[5] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey, “libMesh: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations,” *Eng. Comput.*, vol. 22, no. 3–4, pp. 237–254, 2006.
[6] S. Balay *et al.*, “PETSc/TAO Users Manual,” Argonne National Laboratory, ANL-21/39-Revision 3.23, 2025. doi: 10.2172/2565610.
[7] B. Mouginot *et al.*, *DAGMC - Direct Accelerated Geometry Monte Carlo Toolkit*. (Jan. 2021). [Online]. Available: https://github.com/svalinn/DAGMC
[8] B. Harvey, “Quality Assurance Program Plan (QAPP), Revision 16.” Argonne National Laboratory, Oct. 01, 2024.
[9] “Quality Assurance,” U.S. Department of Energy, Washington, D.C., Order DOE O 414.1D, Sep. 2020.
[10] “MOOSE and MOOSE-Based Applications: Software Quality Assurance Plan (SQAP),” Idaho National Laboratory, PLN-4005, Revision 10, Mar. 2025.
[11] issuing body. American Society of Mechanical Engineers and A. N. S. Institute, *Quality assurance requirements for nuclear facility applications*. New York, N.Y: The American Society of Mechanical Engineers, 2022.
[12] “Quality Manual,” Argonne National Laboratory, LMS-MNL-7, Rev. 8, Sep. 2025.
[13] “SCITECH-8 Material Published Externally.” Argonne National Laboratory, Oct. 19, 2023.
[14] A. E. Slaughter, C. J.Permann, J. M. Miller, B. K. Alger, and S. R. Novascone, “Continuous Integration, In-Code Documentation, and Automation for Nuclear Quality Assurance Conformance,” *Nucl. Technol.*, vol. 0, no. 0, pp. 1–8, 2021, doi: 10.1080/00295450.2020.1826804.
[15] “Procurement: Acquiring Goods and Services: For Argonne Staff.” Argonne National Laboratory, Jan. 29, 2025.

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. Definitions are derived from [10] unless otherwise noted.

baseline: A specification or product 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. [11]

change control: An element of configuration management consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items (configuration items see definition) after formal establishment of their configuration identification [11].

change control board (CCB): The group of individuals who are qualified to act as an independent reviewers (see definition). CCB members are responsible for evaluating and approving or disapproving change requests (see definition) (see Section 9.4.1). Adding individuals to the CCB requires a formal training course and approval of the software managers.

change request (CR): A proposed change to a configuration item, including configuration items stored in the Cardinal repository or elsewhere. A CR that proposes a change to a configuration item stored in the Cardinal repository (e.g., to report a defect or request an enhancement) is equivalent to a pull request (PR) on GitHub.

Continuous Integration, Verification, Enhancement, and Testing (CIVET): Idaho National Laboratory managed support software for automating the build and testing of Cardinal within the MOOSE ecosystem [14].

configuration identification: An element of configuration management, consisting of selecting the configuration items (see definition) 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.

configuration management: A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item (see definition), control changes to those characteristics, record and report change processing and implementation status and verify compliance with specified requirements.

contributor: An individual, including external users, who contributes modifications to Cardinal via a CR or to documentation that is reviewed and accepted by a member of the Cardinal CCB; or an individual, including external users, who reports a defect. All developers and CCB members are contributors. (definition adapted from [10])

defect: Anything observed in the documentation or operation of Cardinal that deviates from expectations based on previously verified software products or reference documents. (see *anomaly* in [10])

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

enhancement: An improvement or enhancement to Cardinal not associated with a defect.

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

independent reviewer: A member of the CCB who reviews proposed changes to the repository in a CR 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 contributor who initiated the CR content.

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

MOOSE: The Multiphysics Object Oriented Simulation Environment, which is custom-developed open source software that provides a library of functions that can be used within other custom-developed software applications for the purpose of mathematical modeling and simulation of engineering applications.

MOOSE-based application: Custom-developed software that derives primary functionality from MOOSE. In this context, Cardinal is a MOOSE-based application.

nonrecord: Informational 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.

open source: Denoting software for which the source code is made freely available and may be redistributed and modified.

Publication Approval Notification and Distribution Application (PANDA): The ANL system used for reviewing, managing, and approving the distribution of scientific and technical information.

pull request (PR): Mechanism for a contributor to propose a change to a Cardinal configuration item stored in the Cardinal repository, using GitHub. Note that in this context, a pull request is equivalent to a change request (CR).

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.

quality level (QL): the grade of quality assigned to the software based on the graded approach defined in the ANL Quality Manual. The grade is determined by the probability of and severity of consequences of a failure. ANL defines four QLs: QL-A, B, C, and D. QL-D is considered non-safety

related, whereas QL-A/B/C describe software related to safety, security, and environment. The QL is determined through ANL review. [12]

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 Cardinal that has been developed according to this plan, has been subjected to verification (see definition) and validation (see definition) through a release review (see definition), and includes a release log.

release log: Documentation associated with a release that provides the baseline (see definition) 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 (see definition) for a release.

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

repository documentation: Documentation considered as a configuration item that is managed as a part of the repository on GitHub.

revision: A stable snapshot of Cardinal that has been managed by the CCB 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, or open source/zero-dollar acquisitions.
- ***custom-developed software:*** Software built at ANL 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, functions, 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 (SQA): All actions that provide adequate confidence that software quality is achieved.

static documentation: Documentation considered as a configuration item that is managed via PANDA or other official ANL systems according to SCITECH-8.

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, predicted results, and a set of execution conditions for a test item. A test case can include:

- ***regression test:*** A test case to verify that modifications have not caused unintended effects and that the system or component still complies with associated requirements.
- ***expected-error test:*** A test case to verify that modifications have not caused unintended effects to expected warning and/or error messages.
- ***validation test:*** A test case that compares the results of the software against a measurement or reference solution. Validation tests are not associated with requirements but are used to verify that the system or component still produces acceptable results.

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

test results: A complete set of results obtained by executing the test cases (see definition) for Cardinal.

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.

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).

3.2 Acronyms

ANL	Argonne National Laboratory
ASME	American Society for Mechanical Engineers

CCB	change control board
CCI	communication and contact information
CIVET	Continuous Integration, Verification, Enhancement and Testing
CM	configuration management
CR	change request
DOE	U.S. Department of Energy
FAR	failure analysis report
IEC	International Electrotechnical Commission
INL	Idaho National Laboratory
ISO	International Organization for Standardization
IT	information technology
M&O	maintenance and operations
MOOSE	Multiphysics Object Oriented Simulation Environment
NQA	Nuclear Quality Assurance
PANDA	Publication Approval Notification and Distribution Application
PR	pull request
QA	quality assurance
QL	quality level
QLD	quality-level determination
RTM	requirements traceability matrix
SCS	software coding standards
SDD	software design description
SLL	software library list
SRS	software requirements specification
SQA	software quality assurance
SQAP	software quality assurance plan

SVVP	software verification and validation plan
TMS	training management system

4 Management

4.1 Organization

As a DOE-sponsored code, Cardinal project planning activities occur annually as part of DOE NEAMS Program work package development efforts. During the planning phase, the Software Managers prepare a baseline work package, schedule, resource allocation, and budget estimate. As per DOE requirements, progress on Cardinal project activities is reported in annual milestone reports, or a similar deliverable determined at the time of project planning. All work package activities that involve modification of Cardinal occur in accordance with the practices, procedures, and requirements set forth in the Cardinal SQA Program, and quality planning is integrated with project planning.

4.2 Roles and Responsibilities

Details on SQA Program roles are provided in Table 4.1. Program personnel have access to the NSE Division QA Representative (QAR), who has informal oversight of the SQA Program.

Table 4.1: Roles and Responsibilities

Role	Tasks and Responsibilities
Software Managers	• Interface with line management to prepare budgets, ensure availability of adequate staffing, create task assignments, and oversee project tasks to ensure that work is carried out on schedule and within budget. • Reassign role and responsibility in this plan, as needed. • Schedule work activities as needed, see Section 9.1.1. • Manage schedule and scope, see Section 9.1.1. • Establish and control business requirements. • Establish schedule for CR as needed, see Section 9.8.2. • Ensure the implementation of the required SQA and training, see Section 13. • Determine correct approach for obtaining error information from a supplier, see Section 8. • Update and maintain project level risks, annually, see Section **Error! Reference source not found.**. • Align software requirements with work, see Section 9.2. • Coordinate and manage software releases, see Section 9.6. • Establish members of the CCB. • Establish timeline for completion of CR, as needed, see Section 9.8.2. • Assess adequacy of change control process, as needed, see Section 9.8.2. • Identify system software (see definition) used for testing and release, see Section 11.2. • Manage project baseline, see Section 9.8.2. • Document and control technical requirements/design per this plan. • Acquire IT materials and services in accordance with ANL 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 users 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 records per this plan, see Section 12. • Create and manage retirement plan.
Software Quality Assurance Lead	• Ensure the implementation of the required SQA and training, see Section 13. • Document and maintain this plan (SQAP) and QLD, see Section 7.2. • Authenticate records, see Section 12.1. • Identify and document the quality level in the Quality Level Determination (QLD). • Review the QLD and revise as needed.
Change Control Board (CCB) Member	• Manage configuration items within the repository and software libraries, see Section 7.2. • Manage modifications to the software, see Section 9. • Oversee problem reporting, see Section 9.8.1. • Act as independent reviewer, see Section 6.2.1. • 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 Section 5.2. • Close CRs. • Document CRs including change descriptions and submit them to the CCB for approval. • Approve or disapprove CR and inform requestor. • If change impacts technical requirements/design, approve revised technical requirements/design. • Collaborate, as necessary, with other CCB members on any of the above responsibilities.

4.3 External Interfaces

Due to its nature as open source software, Cardinal is available to all, including, but not limited to, government, private, and academic scientific research organizations. Access to the repository and other selected digital resources are unlimited.

5 Documentation

Three classes of documentation are defined for Cardinal: static documentation (see definition), repository documentation (see definition), and development documentation (see definition). At a minimum, the documentation listed in the following sections are required. These documents are managed as records in accordance with Section 12.

5.1 Static Documentation

Static documentation is generated at plan initiation using Argonne templates and managed via ANL tools (e.g., PANDA or other formal system).

- Software Quality Assurance Plan (SQAP; this plan), includes the configuration management plan (CMP), software verification and validation plan (SVVP)
- Quality-level determination (QLD) (see Section 9.1.3.1)

5.2 Repository Documentation

The following documents are managed within the repository.

- User documentation (see definition) (e.g. user and/or theory manual, within-code documentation)
- Software requirement specification (SRS)
- Software design description (SDD)
- Requirement traceability matrix (RTM)
- Software test plan (STP)
- Failure analysis report (FAR)
- Communication and contact information (CCI)
- Software coding standards (SCS)
- Software library (see definition) list (SLL)

5.3 Development Documentation

The following documents shall be maintained to document the change control process leading to a release.

- CRs
- Test results
- Release logs
- 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 document 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 of the CCB. The reviewer may rely on assistance from any individual, but the final approval or disapproval is the sole responsibility of the independent reviewer. All comments are retained in the GitHub system.

All reviews and approvals will be recorded within the PR using the integrated approval system of GitHub. This includes identification of the independent reviewer of the CCB prior to acceptance of the proposed modification. Any proposed changes to the repository trigger automated testing.

An independent reviewer of the CCB shall conduct a design review for all CRs 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 is considered to be within the scope of the design verification review.

6.2.2 Release Review

A release requires an additional release review, as specified in Section 9.6.2, to ensure compliance with the approved software requirements. Test results from CIVET are considered part of the review.

7 Configuration Management

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

7.1 Identifying Configuration Items

For Cardinal, the configuration items include:

- Static documentation as defined in Section 5.1.
- Configuration items within the repository, including 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 CR via a comment.

7.2 Managing Configuration Items

Static documentation shall be managed by the SQA Lead 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 SQA Lead, Software Managers, and the relevant QAR. This plan is retained as a record in PANDA.
- The QLD shall be reviewed at the same time the SQAP is reviewed. Modifications to the QLD (see Section 9.1.3.1) require a review and approval process. The QLD is retained as a record in official ANL systems.

The configuration items within the repository will be managed by the CCB and adhere to the change control process detailed in Section 9.8.2 and maintained under CM until the software is retired.

Software libraries will be managed by the CCB and adhere to the change control process detailed in Section 9.8.2. The software libraries required for Cardinal shall be listed in the Software Library List.

7.3 Naming Configuration Items

Static documentation shall follow the naming convention “Cardinal-DocAcronym-rN”, where DocAcronym is the acronym of the documents title, and N is the revision number. Static 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 version control system.

Support software and 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 of Cardinal 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 the CRs. A CR remains in proposed status until it has been approved by an independent reviewer of the CCB or closed. All CRs are tracked for the full duration of their lifetimes, irrespective of status, using the GitHub system for PRs.

A PR can have one of the following statuses:

- Proposed: A PR that is under development by a contributor and reviewed by an independent reviewer. Within GitHub, all PRs that are "open" have this status.
- Approved: A PR that has been reviewed as detailed in Section 9.4.1 and approved by an independent reviewer of the CCB using GitHub approval system. The changes may be merged by any member of the CCB after approval.
- Completed: An approved PR that has been merged.
- Disapproved: A PR that no longer has a proposed status and has not been merged. Within GitHub all PRs 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 Cardinal will either be procured with support from the ANL procurement office in accordance with LMS-MNL-23 [15]. All zero-dollar (no fee) or open source acquisitions will be handled in accordance with LMS-MNL-23. Acquired software necessary for Cardinal 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 team's 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).

When acquiring software (including software libraries) that will be integrated with Cardinal, the following will be incorporated into the life-cycle documentation for that application:

1. Requirements describing the capabilities and limitations of the acquired software.
2. Test cases that will be used to validate the capability of the acquired software.
3. Instructions for use, as applicable.

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.

A software library required for Cardinal shall be listed in the Software Library List (SLL).

9 Software Engineering Method

Cardinal uses test driven development (see definition), an agile software development methodology, with a constant stream of new builds, updates, and modifications to the software that is managed by the CCB. The process includes creating continuous builds that are fully tested before being integrated into the repository. Each stable integration is considered a revision of Cardinal. Only a revision that has followed the release process (see Section 9.6) will be set as a baseline.

All revisions and releases are stored using GitHub and are available for download from this location at any time.

The tasks delineated in the following subsections encompass SQA activities performed throughout the life cycle of Cardinal.

9.1 Project Initiation

Cardinal shall follow the requirements and processes as defined in this plan. If any processes deviate from this plan, an amendment to this SQAP shall be created to document those deviations. As much as possible, the processes for Cardinal shall remain consistent with MOOSE and be repeatable.

9.1.1 Work Activities and Schedule Allocation

Work activities will be defined and delineated for the Cardinal software by the software managers 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 managers.

9.1.2 Resource Allocation

The software managers will allocate resources in coordination with line management. Each resource will participate in the assigned work activities described in Table 4.1. The software managers have the discretion to allocate resources as needed throughout the duration of the project.

Subcontractors may be used. The software managers are responsible for securing funding for the procurement of their services and determining tasks that are subcontracted.

9.1.3 Budget Allocation

The budget for the development of Cardinal 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 on of the software managers.

9.1.3.1 Quality-Level Determination

The QLD documents the decision basis as to why a software application is a specific quality level. The record copy is maintained within official Argonne systems.

9.2 Software Requirements

The requirements for Cardinal align with the work activities defined by software managers (see Section 9.1.1). The requirements shall be recorded within the repository, defined within test cases, and available in the SRS.

The software requirements or other life-cycle documentation shall identify operating systems, functions, interfaces, performance requirements, installation considerations, design inputs, and any design constraints of the computer program; specify technical and software-engineering requirements; identify applicable reference drawings, specifications, codes, standards, regulations, procedures, or instructions that establish software design requirement test, inspection and acceptance criteria, as applicable. Security features (e.g., vulnerability protection, and cybersecurity) shall be specified by the end user, commensurate with the risk from unauthorized access or use.

9.2.1 Requirements Traceability Matrix

Traceability of each requirement through design and testing for the release will be established and maintained within the test cases and available in the RTM. The complete set of test results shall be accessible through the RTM.

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 and referenced within test cases.

9.4 Software Implementation

Contributors utilize the software design to implement the relevant features 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 CR.

9.4.1 Independent Review

Independence is achieved through a variety of measures that are integrated in the development and change control process. Each PR must be peer reviewed by at least one independent reviewer of the CCB. If the PR was initiated by a member of the CCB, the independent reviewer must be a different CCB member. 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 completed satisfactorily, and

the software is approved to follow the release process. However, a release will only be performed if deemed necessary by the software managers. The review documentation is maintained on GitHub.

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 to be performed by an independent reviewer, in addition to the automated testing performed using GitHub Runners and CIVET [14].

9.5.1 Software Test Plan

9.5.1.1 Test Cases

As part of a PR, contributors are expected to add at least one test case that covers implemented code changes, if applicable. An independent reviewer of the CCB must approve a PR and ensure that the PR includes adequate test cases, including the necessary requirements and design associated with newly created or modified test cases. The process for handling PRs that modify requirements is defined in Section 9.8.2.

Test 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 unintended functions nor degrade other software applications running in the operating environment. Review of the online user guide on the general syntax required for Cardinal is sufficient background for the creation of the test case.

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 PR that implemented the test case.
    - The revision when it was created.

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

9.5.2 Software Test Execution

Testing is performed automatically by a GitHub continuous integration (CI)/continuous delivery (CD) Runner and CIVET [14] as part of the change control process when a contributor creates a PR. Table 9.1 summarizes the various levels of testing that occur during the change-control process.

CIVET automated testing will indicate successful execution of test cases. This ensures that Cardinal demonstrates adherence to the documented requirements and that the software produces correct results.

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

Step	Description	Who	What
1	PR Testing	Automated (CIVET)	This testing is executed when a contributor submits a PR through GitHub. The proposed change is checked for adherence to coding standards, compiled, and tested. This step also evaluates the compatibility of Cardinal in the MOOSE ecosystem. The test results are reported as comments and status updates on the PR within GitHub. 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 of the CCB, as detailed in Section 9.4.1. After approval the PR is merged into the development branch ("devel") of the repository.
3	Development Branch Testing	Automated (CIVET)	This step reevaluates test cases to ensure that merged PRs are compatible. This step also evaluates test cases. If successful, the changes are automatically merged to the stable branch. If not, a new PR must be created to correct the problem. The merge to the stable branch ("master") results in a revision.
4	Documentation	Automated (GitHub Runner)/Software Manager	The Cardinal documentation, which is automatically compiled for the new revision, is made available by one of the software managers by pushing to the website with each release, or sooner if deemed necessary.
5	Release	Software Managers	As necessary, the release process will be followed to establish baseline as described in Section 9.6.

9.5.3 Test Results

Test results are accessible from the RTM 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 PR and is performed by an independent reviewer of the CCB. It entails viewing the test results provided by CIVET. Because each test case defines a requirement, if all test cases pass then all requirements have been satisfied. This removes the burden of having to determine if test cases meet acceptability requirements.

9.5.5 Test Records

The generated test reports are maintained as test records with each release as outlined in Section 9.6.3.

9.6 Version Release

Software intended for use in a quality setting must undergo the release process detailed in this section. The releases of Cardinal are supported for at least the duration between one release and the next.

Revisions present in the repository, as detailed in Table 9.1, are not intended to be used in a safety or quality setting. The revision is provided "as-is," and no guarantee of functionality or performance of a revision is provided. Hence, a release of Cardinal must depend on a release of MOOSE.

ANL is not a qualified supplier of safety software. As such, software released in accordance with this section requires the end user to perform additional end-use qualifications and/or dedication prior to the software being used in a safety or quality setting.

9.6.1 Release Candidate Identification and Control

Any revision can be selected as a candidate for release for a quality setting.

9.6.2 Release Review

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

One of the software managers 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 items in the repository and (2) software libraries. The ability to list the configuration items 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, one of the software managers 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. The branch is tagged with the unique identifier for the release in accordance with Section 7.3.
- Adding the release log to the labeled branch.
- Adding the associated test results to the labeled branch.

Completion of these operations for Cardinal 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 items 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 that this is a copy of the content that exists in the release and is added for convenience.

Releases may receive patches to resolve defects via an amended release 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 Software Acceptance

It is incumbent upon any organization relying on Cardinal 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, use of Cardinal shall be controlled in accordance with the user organizations approved procedures and instructions.

9.8.1 Problem Reporting and Corrective Action

Any user can report a problem. A problem should be reported in one of three ways as detailed in the CCI (see Section 9.8.3):

- Direct communication
- Creation of an issue
- Creation of a PR (the GitHub implementation for CRs)

In the event that the problem is reported as a direct communication to the CCB, a problem will be evaluated 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 CCB (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, an issue will be created by a member of the CCB (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 correction actions within the scope of Cardinal are tracked using the change-control process, as defined in Section 9.8.2. Defect resolution may be tracked via the original PR related to the modification that led to the defect.

9.8.2 Software Change Control

All configuration items 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 (CR), also known as a pull request (PR), 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 managers to work on existing approved issues that are viewable on GitHub. 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 contributing code to Cardinal shall be included in the CCI. Contributors are encouraged to reach out to the software managers prior to implementation.

9.8.2.1 Initiating Changes

The process for initiating a change to Cardinal is flexible, to accommodate external contributors and encourage the growth of the development community. Thus, changes may be initiated through unsolicited community-driven efforts.

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 pull request (PR).

All issues and PRs shall be evaluated by a member of the CCB 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 GitHub.
- 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 CCB shall ensure that the required information (as detailed below) is added to the issue or PR.

Enhancements

An enhancement shall include the following and shall be recorded in an issue or PR.

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

Defects

All defects are captured by creating an issue if it was not previously created, either by the user or a member of the CCB. 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 items.

A member of the CCB will evaluate the impact of a defect on project resources. 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 “defect” with GitHub.

A member of the CCB will determine the level as defined below, for each issue that is a defect and evaluate the impact on Cardinal.

- Significant. Issues affecting the operation or execution of the code, with a high possibility of significantly affecting the accuracy of the results.
- Normal. Issues affecting the operation or execution of the code, but with a low possibility of significantly affecting the results.
- Minor. Issues that do not affect the accuracy of the results.

The defect resolution will be contained within a PR and be associated with the issue.

9.8.2.2 Implementing Changes

The GitHub PR process is followed for all CRs. This process incorporates the agile software development life cycle, including requirements, design, implementation, testing, and independent review.

Any contributor can choose to work on a particular issue by associating the issue with a PR in GitHub. A listing of all issues is available through GitHub.

If the PR requires code changes, the independent reviewer of the CCB will ensure appropriate test cases, requirements, design documentation, and appropriate cross-referencing are all included within the PR. The tests performed by CIVET as defined in the STP enforce the inclusion of each of these components.

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

9.8.2.3 Cursory Evaluation

Once the CR is implemented, CIVET begins the process of running checks on the proposed changes. Namely to the extent possible the coding standards are enforced via code style checks, proper issue refencing, spacing, size checks, etc. as defined in the STP. These are performed to provide expeditious 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 PR. It is important to note that not every approved PR results in a release. The PR process in Steps 1-4 in Table 9.1 can 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 PR.

The independent reviewer controls and is responsible for the evaluation and disposition of proposed changes for all configuration items 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 PR. 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 PR for communications.

There is no established time for review of the PRs; however, consideration should be given to the priority established by the CCB.

9.8.2.5 Approving or Disapproving Changes

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

definitions in Section 7.5), the name of the CCB member giving the final disposition, and the date of the disposition shall be recorded (this is automatic within GitHub).

If approved, the proposed changes associated with the PR are merged into the development branch ("devel") of Cardinal. 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 of Cardinal shall be specified in the CCI.

The CCI shall include:

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

9.9 Software Retirement

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

The software managers shall determine when support of Cardinal 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 records disposition schedule.
- The retired Cardinal will be archived within the GitHub infrastructure to allow access "as-is" in a read-only state and disallow any future contributions or changes to configuration items.
- All affected websites and links to the retired software shall be updated to reflect the status of the software.
- Support will be discontinued, and an announcement will be posted to the Cardinal website and repository to serve as notification to users.

10 Standards, Practices, Conventions, and Metrics

10.1 Software Coding Standards

Any new software changes and developments in Cardinal shall follow a coding standard on all source code within the repository. The coding standards shall be detailed in the SCS. The independent reviewer of a CR shall ensure that the applicable coding standards has been followed before the proposed change(s) are merged.

10.2 Methods, Techniques, and Tools

As Cardinal, a MOOSE-based application, continuously evolves, it is necessary to use modern software-development tools, methodologies, and techniques to maintain its usefulness to the user community. Methods, techniques, and tools are identified below.

A test-driven development process is used through pull requests (PRs) as described in Section 9. Within this process, various techniques – including code coverage analysis; regression, error, and validation 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, GitHub repository services, and a range of Python-based tools for testing and documentation.

11 Support Software

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

11.1 GitHub Repository

The GitHub 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 managers 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 using GitHub runners and CIVET to ensure that requirements for Cardinal are satisfied.

11.3 Software Tools

The following software tools are used for obtaining test results for a release. The tools used are documented in the test results. These tools follow the procedures detailed in the MOOSE SQAP in the same capacity as a MOOSE-based application, with the following exceptions.

- The repository documentation associated with the tools are limited to the RTM and SDD.
- A release of the tools is not required. Cardinal uses a revision of the tools as noted in the release log.
- The reviewer is not required to be independent and the review may be conducted by the same member of the CCB who initiated the PR.
- PRs initiated by a contributor who is not a member of the CCB require a review by a member of the CCB.

11.3.1 Continuous Integration Verification Enhancement and Testing (CIVET)

CIVET provides a client/server continuous integration testing services for MOOSE-based applications. CIVET is able to integrate with GitHub repository services. CIVET maintains a database of records and versions all changes to testing recipes, making it ideal for working with configuration items. CIVET is maintained as a public repository on GitHub. The version of CIVET used for a release is recorded in the release log (see Section 9.6.2). [14]

11.3.2 MOOSE Tools

The testing and documentation software used by MOOSE and Cardinal is custom-developed software and is maintained as part of the public MOOSE repository on GitHub. These tools are used by CIVET to run the test cases and obtain test results as well as build the repository documentation. Therefore, any change to the use of the testing or documentation with respect to integrating with Cardinal follows the change-control process detailed in this document.

12 Records Collection, Maintenance, and Retention

The repository and development quality records identified in Section 5 will be held within GitHub repositories, while the static documentation will be maintained via official Argonne systems (PANDA or otherwise).

Artifacts related to Cardinal that are produced during the software development life cycle meet the definition of records and do not meet the definition of nonrecords. Therefore, everything produced shall be managed as a record, and segregation of records from nonrecords is not applicable to this plan.

12.1 Records Authentication

Quality records are authenticated by the SQA lead by verifying that all documents identified in Section 5 are accurate and complete in accordance with Section 9.6.3. For records retained in GitHub, the PR process described in Section 9.8.2 includes the approval cycle and the control points where approval signatures are captured.

13 Training

The software managers, in coordination with line management, are responsible in ensuring that personnel who work on Cardinal are qualified and trained in accordance with LMS-PROC-16 "Mandatory Training." Each role identified in Table 4.1 in this plan shall complete the training identified in Table 13.1.

Table 13.1 Training/Qualification Requirements by Role

Role*	Course: EQO203	Plan: Cardinal-SQAP Required Read	Course: RDHERD01	Course: 0INL1862	Cardinal CCB Training
Software Managers	Required	Required	Required	Required	N/A
SQA Lead	Required	Required	Required	Required	Required
CCB Member	N/A	Required	Required	N/A	Required
	* See Table 4.1 for a mapping of responsibilities back to roles in this document. Note that one individual may hold more than one role (e.g. Software managers may also be a CCB member).				

- EQO203: Quality Assurance Program Plan (QAPP) General Training: This course presents an outline of the Quality Assurance Program Plan (QAPP) to provide awareness of the DOE approved institutional QA Program for ANL.

- Cardinal-SQAP Required Read: A reading of this software quality assurance plan (SQAP) is required for all roles to ensure Cardinal team members and management gain a basic understanding of the processes that ensure that software quality is embedded as software is developed. Compliance to this SQAP is obligatory throughout the software life cycle, including planning, requirements, acquisition, design, implementation, acceptance testing, maintenance and operations (M&O), and retirement.

- RDHERD01: Introduction to MOOSE and MOOSE-based Applications Development Processes: Course providing basic information on MOOSE and MOOSE-based applications development processes, including roles and responsibilities, change-control process, and the release approach detailed in the MOOSE SQAP.

- 0INL1862R: MOOSE and MOOSE-based Applications Project Lead and SQA Lead Training: Course providing detailed information for the project leads on the MOOSE and MOOSE-based applications development processes included roles and responsibilities, change control process and the release approached detailed in the MOOSE SQAP. Course also includes detailed information for the SQA lead related to their specific roles and responsibilities.

- Cardinal Change Control Board (CCB) training: training to review the SQAP (this document)

and all related SQA documents and procedures to ensure understanding of responsibilities.

Courses that are administered via TMS will retain record of completion for ANL employees. For the INL/MOOSE trainings, personnel shall inform the SQA lead and one of the software managers in writing that they have read and understood the material. Personnel who attend the CCB training will be recorded on an attendance list that is maintained by the SQA lead.

CCB 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 managers. This list shall be reviewed and updated with at least each release, or as deemed necessary by the software managers.

When this plan changes, the software managers and SQA lead will evaluate the extent of the change and determine if additional or re-taking of any training is required. Training will be developed and assigned accordingly.

www.anl.gov

ARGONNE NATIONAL LABORATORY

- U.S. Department of Energy research facility
- Operated by the UChicago Argonne, LLC
- Midwest's largest federally funded R&D facility
- Located in Lemont, IL, about 25 miles (40 km) southwest of Chicago, IL (USA)
- Conducts basic and applied research in dozens of fields
- Unique suite of leading-edge and rare scientific user facilities

CONTACT
Argonne National Laboratory
9700 South Cass Avenue
Lemont, IL 60439
630-252-2000

U.S. DEPARTMENT of ENERGY

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