INL/EXT-18-45020

NEAMS-IPL MOOSE Midyear Framework Activities

Andrew Slaughter
Cody Permann
Derek Gaston
Richard Martineau

March 2018

The INL is a U.S. Department of Energy National Laboratory operated by Battelle Energy Alliance

INL/EXT-18-45020

NEAMS-IPL MOOSE
Midyear Framework Activities

Andrew Slaughter
Cody Permann
Derek Gaston
Richard Martineau

March 2018

Idaho National Laboratory
Modeling and Simulation Department
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

ABSTRACT

Multiphysics Object Oriented Simulation Environement (MOOSE) is a modular, pluggable framework for building complex simulations. The ability to add new objects with custom syntax is a core capability that makes MOOSE a powerful platform for coupling multiple applications together within a single environment. Documentation is not only a required component for a complete software quality assurance plan, but is critical to the overall success and wide-spread adoption of the platform by other researchers and institutions. Therefore, the notion of a modular, pluggable framework has been extended to the creation of documentation content. This report details the methodology for defining the NQA-1 compliant requirements in a manner that allows for the automatic generation of the Software Requirement Specification (SRS), Software Design Description (SDD), and Requirement Traceabiltity Matrix (RTM) from any repository revision.

ACRONYMS

ACM Association for Computing Machinery

IEEE Institute of Electrical and Electronics Engineers

MOOSE Multiphysics Object Oriented Simulation Environement

NQA-1 Nuclear Quality Assurance-1

RTM Requirement Traceabiltity Matrix

SDD Software Design Description

SQA Software Quality Assurance

SRS Software Requirement Specification

1 Automated Software Design Document

Multiphysics Object Oriented Simulation Environement (MOOSE) is designed to be a modular, pluggable framework for building complex simulations. The core of this design relies on object-oriented programming paradigms. These paradigms have been expanded to the creation of documentation content as well. In particular, it is possible to create Nuclear Quality Assurance-1 (NQA-1) documents, such as the Software Requirement Specification (SRS), Software Design Description (SDD), and Requirement Traceabiltity Matrix (RTM), using templates that reside within the MOOSE repository. As a result, MOOSE-based applications can leverage these capabilities to partially fulfill Software Quality Assurance (SQA) compliance at an accelerated rate. Application developers are encouraged to follow the MOOSE development process by filling out templates for building the necessary content. This report focuses on two aspects of the NQA-1 documents:

1. the definition of requirements and
2. the automatic generation of NQA-1 documents using the requirement definitions.

In short, if requirements, design, and other SQA related attributes are defined as done within MOOSE then it is possible to automatically build significant portions of the SRS, SDD, and RTM documents.

1.1 Requirement Definitions

NQA-1 requirements, in particular the functional requirement, must be:

- correct,
- unambiguous,
- complete,
- consistent,
- verifiable, and
- traceable.

In general, a requirement is defined within the SRS; each requirement is then linked to a test within the RTM; finally, the SDD links design to the requirement. Creating and maintaining this set of documents is an enormous burden and hinders the ability to rapidly create, deploy, and maintain software that follows the NQA-1 process. Mitigating this burden and integrating definition of system requirements into the regular development process is the driver behind the work presented herein.

With respect to MOOSE, the above definition of a requirement describes each test within the framework, with two exceptions. First, the requirement wording does not exist for a test. Second, a test is not "traceable", where traceable in this context refers to the ability to connect a code change request to the software design and associated requirement.

Each test within MOOSE is invoked via a test specification. This specification defines the type of test, the settings to use, and the expected result. When three additional items are added to a test specification—"requirement", "issues", and "design"—it fits the definition of a requirement. The following lists details the additional items that are added to test specifications.

- **requirement**: Text that describes the requirement that the associated test satisfies.
- **issues**: List of issue numbers that motivated the creation of the requirement. For MOOSE issue numbers are GitHub issues that contain user stories and links to the associated code change requests.
- **design**: List of MooseDocs[1] pages that contain the design documentation for the system(s) being tested.

The following is a snippet from MOOSE that contains a test specification that includes the "requirement", "issues", and "design" items.

```
[./mark_and_adapt]
  type = 'Exodiff'
  input = 'box_marker_adapt_test.i'
  exodiff = 'box_marker_adapt_test_out.e-s002'
  scale_refine = 2
  requirement = "The adaptivity system shall adapt the mesh within a rectangular region."
  design = "/Markers/index.md /BoxMarker.md"
  issues = '#1275'
[../]
```

Adding these items to test specifications obviously requires effort. However, while the creation of tests is a regular occurrence for developers, it requires minimal effort compared to the work that is associated with code changes to satisfy the test. These items are also testable themselves, thus it is possible to automatically check for the existence of these items such that no code entering the repository is allowed without these items and the associated design documentation content, making the maintenance of the requirements an incremental, rather than monolithic, process.

1.2 Automatic Documentation Generation

MooseDocs is the internal documentation system developed for MOOSE and MOOSE-based applications that allows for websites (html), as well as other formats, to be created from a common language. The system is also customizable such that custom commands may be created that generate content automatically. For this report that content is the various NQA-1 documentation associated with requirements within the SRS, SDD, and RTM.

Each SQA document begins as a template based upon industry standard sources such as Institute of Electrical and Electronics Engineers (IEEE), or the Association for Computing Machinery (ACM). The INL has it's own templates based upon these standards with additional requirements. These templates can be easily converted to markdown syntax and processed by the MooseDocs system. These templates contain the required structure and basic content but then specific language related to the actual project is inserted and serves as the overview or other generally applicable portions of the document. The remaining content is then automatically found in test specification files (See 1.1), processed, and substituted into the relevant document automatically by MooseDocs. This linkage happens through a single statement added to the document, as shown below. Figure 1 is an example of the resulting content from this command.[2]

```
!sqa requirements link=False
```

[1] MooseDocs is the internal documentation system used by MOOSE.

[2] The complete SRS is available at `http://www.mooseframework.org/moose/documentation/sqa/moose_srs.html`.

The RTM is created in similar fashion with the following command.

```
!sqa requirements link=True
```

An example of the resulting output is shown in Figure 2. The important difference is that the RTM includes links to the test specification (i.e., the test) that satisfies the requirement, links to the associated design documentation, and links to the associated GitHub issue that includes the user stories and code change requests.[3]

Finally, the requirement cross-referencing between the design and requirements is generated within SDD with the following MooseDocs command.

```
!sqa requirements link=True
```

Figure 3 provides an example of the automatically generated cross-reference list.[4] The requirement cross-reference contains all the same content as presented in the RTM, but it is organized with the design documentation.

2 Conclusion

Creating and maintaining up-to-date software documentation is a difficult task. The additional requirement of maintaining proper NQA-1 documentation within a highly active software project is nearly impossible without some amount of automation and control so that individual developers can contribute incremental changes over time. Annotating test specifications within MOOSE to include requirement, traceability, and design information allows for the documentation process to be a part of regular development as well as creates a means to automatically generate the various requirement related items within the SRS, RTM, and SDD. This approach is extendable to all MOOSE-based applications, which lessens the effort required for applications to meet the NQA-1 standards.

[3]The complete RTM is available at `http://www.mooseframework.org/moose/documentation/sqa/moose_rtm.html`.

[4]The complete RTM is available at `http://www.mooseframework.org/moose/documentation/sqa/moose_sdd.html`.

Functional Requirements

InterfaceKernel Objects	
F1.1	The InterfaceKernel system shall operate with coupled variables in 2D.
F1.2	The Jacobian evaluation for InterfaceKernel objects in 2D shall be analytically correct.
F1.3	The finite difference preconditioner shall work in parallel.
F1.4	Adaptivity shall work with interface kernels as long as stateful properties are not used.
F1.5	The system shall error if the triad of interface kernels, adaptivity, and stateful properties are used together.

Mesh	
F2.1	A mesh can be split into a specified number of files using command line options.
F2.2	A mesh can be pre-split properly and used to generate equivalent results to running a simulation with the unsplit mesh.

Figure 1: Example of automatically generated list of functional requirement for the MOOSE SRS.

Requirements Traceability Matrix

InterfaceKernel Objects

F1.1 The InterfaceKernel system shall operate with coupled variables in 2D.
Specification: interfacekernels/2d_interface:test
Design: InterfaceKernels System
Issues: #7885

F1.2 The Jacobian evaluation for InterfaceKernel objects in 2D shall be analytically correct.
Specification: interfacekernels/2d_interface:jacobian_test
Design: InterfaceKernels System
Issues: #7437

F1.3 The finite difference preconditioner shall work in parallel.
Specification: interfacekernels/2d_interface:parallel_fdp_test
Design: FDP
Issues: #10375

F1.4 Adaptivity shall work with interface kernels as long as stateful properties are not used.
Specification: interfacekernels/adaptivity:ik_adaptivity
Design: InterfaceKernels System
Issues: #10977

F1.5 The system shall error if the triad of interface kernels, adaptivity, and stateful properties are used together.
Specification: interfacekernels/adaptivity:error_stateful_ik_adaptivity
Design: InterfaceKernels System
Issues: #10977

Mesh

F2.1 A mesh can be split into a specified number of files using command line options.
Specification: mesh/splitting:make_split
Design: Generating Split Configurations
Issues: #10623

F2.2 A mesh can be pre-split properly and used to generate equivalent results to running a simulation with the unsplit mesh.
Specification: mesh/splitting:use_split
Design: Generating Split Configurations
Issues: #10623

Figure 2: Example of automatically generated list of functional requirement for MOOSE RTM.

Requirements Cross-Reference

Markers System

F3.1 The adaptivity system shall create an auxiliary field variable that marks elements for refinement within a rectangular region.
Specification: markers/box_marker:mark_only
Design: Markers SystemBoxMarker
Issues: #1275

F3.2 The adaptivity system shall adapt the mesh within a rectangular region.
Specification: markers/box_marker:mark_and_adapt
Design: Markers SystemBoxMarker
Issues: #1275

F5.1 The system shall support marking elements for refinement during initial setup using a different marker than used during execution.
Specification: adaptivity/initial_marker:test
Design: Markers System
Issues: #1700

InterfaceKernels System

F1.1 The InterfaceKernel system shall operate with coupled variables in 2D.
Specification: interfacekernels/2d_interface:test
Design: InterfaceKernels System
Issues: #7885

F1.2 The Jacobian evaluation for InterfaceKernel objects in 2D shall be analytically correct.
Specification: interfacekernels/2d_interface:jacobian_test
Design: InterfaceKernels System
Issues: #7437

F1.4 Adaptivity shall work with interface kernels as long as stateful properties are not used.
Specification: interfacekernels/adaptivity:ik_adaptivity
Design: InterfaceKernels System
Issues: #10977

F1.5 The system shall error if the triad of interface kernels, adaptivity, and stateful properties are used together.
Specification: interfacekernels/adaptivity:error_stateful_ik_adaptivity
Design: InterfaceKernels System
Issues: #10977

Figure 3: Example of automatically generated list of functional requirement for MOOSE SDD.