Title
Quality assurance within the PyNE open source toolkit

Permalink
https://escholarship.org/uc/item/7897p8p3

Authors
Biondo, E
Scopatz, A
Gidden, M
et al.

Publication Date
2014

Copyright Information
This work is made available under the terms of a Creative Commons Attribution License, available at https://creativecommons.org/licenses/by/4.0/

Peer reviewed
INTRODUCTION

In order for PyNE [1] to be broadly adopted by the nuclear science and engineering community, it must conform to rigorous software development standards. In this work we discuss how the philosophy and implementation of PyNE’s development practices are consistent with the American Society of Mechanical Engineers (ASME) NQA-1 standards [2][3], and what actions will need to be taken for full compliance.

PyNE is a trans-institutional, open source project consisting of a collection of computational tools pertinent to nuclear engineering analysis and simulations. The capabilities of PyNE include canonical nuclide and reaction naming conventions, material handling, nuclear data and cross-section reading, mesh operations, and physics-code-specific capabilities. Users can use PyNE code as a component within their own software to simplify complex tasks such as radiation shielding calculations [4].

PyNE is cognizant of exacting regulations that apply to much of the potential user base. Many applications require software that meets the quality assurance (QA) criteria set forth by the U.S. Nuclear Regulatory Commission (NRC). The NRC endorses the ASME NQA-1-2008 regulatory standard Quality Assurance Requirements for Nuclear Facility Applications (Parts I and II) [5] with the NQA-1a-2009 addendum [3] for the design and construction of nuclear power plants and fuel reprocessing facilities [5].

These documents allude to the traditional software development workflow known as the waterfall model [6]. This strategy features several stages: specification (or requirements), design, implementation, verification, and maintenance. Each stage must be completed and reviewed before continuing on to the next stage. The waterfall model has been shown to be very effective for large, technical projects. However, with the advent of distributed version control as well as physically- and organizationally-disparate development teams came the rise in popularity of agile development [7]. The overarching principle of the agile strategy is to have a short iteration cycle so that the software may evolve and respond quickly to changing needs and use cases.

In practice, the agile method is a series of parallel waterfalls for every issue or requirement that arises. All code must be designed, written, reviewed, tested, and verified. This process happens for all subsets of the code base individually. This stands opposed to the waterfall model where the review process happens over the entire code base as an aggregate entity.

Agile methods are useful for teams that are small, physically separate, at different institutions, or unfunded. This is because the overhead is much less than required for waterfall strategies. Agile mechanisms allow for developers to focus on what is interesting or useful to them on an as-needed basis. Toolkits or library projects (rather than simulators or user interfaces) are particularly ripe for agile development. A well-designed toolkit should be modular in nature, which allows for parallel streams of development. PyNE follows an agile development strategy as much out of necessity as out of ideological fit. Regardless of the development strategy, nuclear code must adhere to the highest standards of quality.

Organizations seeking to comply with NQA-1-2008/NQA-1a-2009 can use any portion of PyNE (or code from any source for that matter) as a component of their design and/or operations software by complying with Part II Subpart 2.7 Section 302, “Otherwise Acquired Software.” This section provides provisions for the use of “freeware” that “has not been previously approved under a program consistent with [the NQA-1 standard]” [3]. To facilitate its use, PyNE seeks to develop code that is fully compliant with these standards, freeing users of any verification and validation burden.

This paper presents the strategy implemented by PyNE to ensure that open source, community-developed code is written, reviewed, tested, and documented in a manner compliant with regulatory standards. The requirements of ASME-1-2008/ASME-1a-2009 are addressed explicitly. The process for declaring PyNE code fully compliant is discussed. These efforts make the PyNE code base suitable for a larger subset of the nuclear engineering community.

PYNE SOFTWARE DEVELOPMENT PRACTICES

The PyNE software development workflow is meticulous, systematic, and similar to workflows of prominent and well-established projects such as the Linux kernel. The PyNE workflow is centered around the Git [8] distributed version control software and additional features provided by the GitHub repository hosting service. The develop and master branches of this repository represent a software baseline: a collection of software that has gone through the review and approval process and cannot
be changed without going through an agreed-upon, formal change process. All PyNE code, tests, and documentation are stored in a public repository on GitHub. The PyNE website (http://pyne.io) is home to the rendered documentation, which includes a user’s guide, developer’s guide, style guide, theory manual, and application programming interface (API) documentation. PyNE also has user and developer mailing lists for questions, discussion, and feedback.

In order to make changes (e.g., new features, documentation improvements, and bug fixes), developers must first make changes in their own versions of the repository (a fork in GitHub terminology) and then issue a GitHub pull request accompanied by a description of the changes. A pull request is a request for changes from a developer’s fork to be applied to the main repository. The acceptance of a pull request is contingent upon the successful completion of a formal review process.

In the formal review process, another developer who did not participate in the writing of the changes reviews the changes line-by-line, verifying the changes are (1) consistent with the requester’s description and (2) compliant with the PyNE developer’s guide and style guide. In addition, the reviewer must ensure that all pre-existing unit tests pass and that any new features implemented in the pull request are supported by additional appropriate unit tests. Continuous Integration (CI)—explained below—assists developers in this process. The review process tends to be iterative, where requesters update their pull requests in response to the criticism of the reviewer. Once the reviewer or reviewers are satisfied, the pull request is accepted, and the newest version of the code becomes the baseline. Record of the pull request is stored by GitHub for the lifetime of the repository.

In order to use PyNE code, users may download the code base directly from the GitHub repository, or they may download a stable release version from the PyNE website. Once a user downloads and installs PyNE, all unit tests can be run locally (with no internet connection required) to verify the installation. The user’s local copy is unaffected by any changes to the main PyNE Git repository, unless the user chooses to update their installation.

Nightly Testing and Continuous Integration

PyNE uses a branching development strategy, where the develop branch corresponds to the actively developed and updated code base. With the develop branch changing so frequently, it is important for developers to ensure that all tests pass on this baseline. In addition to human verification by code review, the PyNE development workflow includes both nightly testing and CI to guarantee that the code base is consistently and constantly tested. These practices only apply to the main PyNE repository on GitHub, not a user’s local copy.

The practice of Continuous Integration is a well-known software engineering practice first conceived by Beck and Jeffries [9] that espouses updating shared code bases in a rapid manner in order to streamline the shared development process. A key component of modern CI approaches is the automated building and testing of a code base upon reception of a proposed change (i.e., a patch). For each proposed patch, the entire patched code base is built on one or more platforms (e.g., a flavor of Linux, a Mac Operating System (OS), or a Windows OS), and some number of unit, integration, and regression tests are executed. The quantity of tests run with each patch submission is dependent on the amount of time required for their execution, allowing for relatively quick knowledge of success or failure. The proposed patch is only accepted if building and testing passes on all supported platforms and the reviewer is content with both the patch’s style and content. PyNE enforces a CI approach for any patch that is proposed to be merged into the develop branch.

While CI is designed to streamline the integration process, nightly testing is a complementary tool that tests the code base once each 24-hour period. Each nightly test is run on one or more platforms. On a clean version of each platform, all dependencies are installed, and the entire code base is built and installed. All unit, integration, and regression tests are then run. A common practice among software projects is to update all supported develop binaries after a successful nightly build and test execution.

PyNE’s Implementation

PyNE uses the Build and Test Lab (BaTLab) [10] as its platform for automatically building and testing software. BaTLab provides the basic tools required to install a project and its dependencies, test a project, and return any built products. PyNE builds and tests both the nightly builds and CI using Ubuntu 12.04 and MacOS 10.8, and BaTLab supports many other platforms that could be used by PyNE.

PyNE uses BaTLab’s services directly for nightly testing. Scripts were developed that—each night—install PyNE’s dependencies, download the current version of PyNE’s develop branch, install PyNE, and run all tests in the PyNE code base. Upon completion, the PyNE developer mailing list is notified in case of any failures.

The implementation of CI is more complicated, because it must communicate with PyNE’s repository management service, GitHub, and a build and test service. There exist a number of CI services, such as TravisCI [11]; however, many have limitations in either their number and type of supported platforms or constrained total building and testing time. PyNE uses Polyphemus [12], a plugin-based CI service that can connect with any supported front end (e.g., GitHub) and back end (e.g., BaTLab).

The PyNE development team runs a Polyphemus server
that uses its BaTLab scripts to build and test all proposed patches to PyNE’s develop branch. A patch proposal is initiated via GitHub’s pull request interface. The Polyphemus server is then notified, causing the launch of a BaTLab job. Upon completion of the BaTLab job, the server is again notified and updates GitHub with the result. Finally, active and prior jobs can be monitored via a continuously running dashboard.

ADDRESSING NQA-1-2008/NQA-1A-2009

NQA-1-2008/NQA-1a-2009 contains two sections pertinent to nuclear engineering software development. In Part I: “Requirements for Quality Assurance Programs for Nuclear Facilities,” Requirement 3 Section 800 addresses “Software Design Control” and is unrevised by the 2009 addendum. The second portion is Part II: “Quality Assurance Requirements for Facility Applications”, Subpart 2.7: “Quality Assurance Requirements for Computer Software for Nuclear Applications.” A revised edition to Subpart 2.7 appears in-full in NQA-1a-2009. The software development practices of PyNE can be mapped to the criteria set forth by these documents. The section headings below refer to pertinent items within these documents.

NQA-1-2008 Part I Requirement 3 Section 800: Software Design Control

801 Design Process

The design process suggested by Section 801 is essentially the waterfall method, which requires documentation of all of the following, described in Pars. 801.1-801.5:

1. Requirements - the scope of the capabilities implemented by the work.
2. Software design - the mathematical and computational methodologies employed to meet the requirements.
3. Implementation - writing code using the standards and conventions agreed upon by the organization.
4. Software design verification - independent confirmation that requirements are met.
5. Computer program testing - comparison of results produced by the work to expected or known results.

In PyNE, requirements appear in the description of changes made in a pull request. The software design of features is documented within the function/class documentation strings that become part of the API documentation. Documentation of more sophisticated methods can be provided in the PyNE theory manual as needed. The implementation is uniform throughout PyNE as described in the user’s and developer’s guides. Pull requests constitute software design verification and the extensive use of unit testing, continuous integration, and nightly builds satisfy computer program testing.

802 Software Configuration Management

In addition to documenting the steps of the Waterfall method, documentation must address configuration identification, configuration change control, and configuration status control. Configuration identification requires that each version of the code can be uniquely identified and the difference between versions can be ascertained. In PyNE, all code, tests, and documentation are version controlled with Git, which fully and automatically supports these capabilities. Configuration change control is documentation of the changes to software baselines including the rational for the change and appropriate verification. In PyNE, all changes are cataloged on GitHub in the form of closed (completed) pull requests that remain on the GitHub website for the lifetime of the repository. Configuration status control requires that code can be accounted for prior to being incorporated into the baseline and changes that are “proposed and approved, but not implemented” are documented [3]. This is accomplished by the fact that all changes to the baseline come from pull requests from developer forks, which are version controlled in the same fashion as the baseline and can be viewed publicly on GitHub. GitHub provides an issue feature, which documents pending changes. If users make modifications to their local versions of PyNE, they are responsible for the configuration management associated with the changes.

NQA-1a-2009 Part II: Subpart 2.7

The criteria in this subpart are explicitly stated to be supplementary to that of Part I. Likewise, PyNE’s strategy for complying with Part I Requirement 3 will also satisfy much of Part II Subpart 2.7.

Section 200 General Requirements

Sections 201-203 describe requirements for documentation, code review, and software configuration management. Using PyNE’s strategy, these requirements are redundant with those described in Part I. Section 204 outlines requirements for reporting software bugs and the corrective action that ensues. A GitHub issue is created if bugs are identified, which automatically notifies the development team. GitHub issues also provide a mechanism for assigning bug fixes to the relevant developer(s).

300 Software Acquisition

Section 301: “Procured Software and Acquired Service” does not apply to PyNE, though PyNE does rely on code from other open source projects. Use of this code is contingent upon compliance with Section 302: “Otherwise Acquired Software.” Within PyNE, this must be done on a case-by-case basis. Some portions of PyNE have no
dependencies, while other portions may require code—in varying capacity—from a multitude of sources of various project sizes and types.

400 Software Engineering Method

Sections 401 to 404 describe design requirements, design verification, implementation, and acceptance testing: reiterating the waterfall method. NQA-1a-2009 revises this section to include security requirements. Security requirements, and also the requirements in Section 405: “Operation,” are the responsibility of the end-user and are outside the scope of PyNE.

500 Standards, Conventions, and Other Work Practices.

The documentation of standards, conventions, and work practices required by this section is provided in the PyNE developer’s guide and PyNE style guide.

600 Support Software

This section covers software tools and system software, which is also the responsibility of the end-user.

FULL COMPLIANCE

Compliance with NQA-1-2008/NQA-1a-2009 requires extensive documentation. Though the software development practices of PyNE generally satisfy the criteria outlined by NQA-1-2008/NQA-1a-2009, formal documentation of this is lacking in some cases. This deficiency is acknowledged in the form of Python import warnings. Currently, when a module of Python code from PyNE is imported by another script for use, the user is warned that the code is not fully compliant. As modules are developed to a logical point of completion, necessary documentation will be completed and import warnings will be removed. This documentation will itemize the requirements of NQA-1-2008/NQA-1a-2009, explain how they are met, and will then require independent approval from a member of the PyNE development team.

CONCLUSION

The community-based software development model adopted by PyNE does not preclude its use in applications that require ASME NQA-1-2008/NQA-1a-2009 compliance, which provides provisions for such software. In addition, PyNE is working towards full compliance with these standards. Through the use of version control, pull requests, unit testing, nightly builds, and continuous integration, PyNE software development practices accomplish the goals of meticulous record-keeping, peer-review, validation, and accountability demanded by the industry standard. Additional documentation of these practices for individual modules within PyNE will make them fully compliant. These efforts ensure the accuracy and dependability of the PyNE code base.

ACKNOWLEDGEMENTS

We would like to thank Nicholas Touran for the valuable advice he provided, which guided many of the efforts described in this paper.

REFERENCES