Useful Practices in Open-Source Software Development for Nuclear Science and Engineering

Oleksandr Yardas
Advanced Reactors and Fuel Cycles Group

University of Illinois at Urbana-Champaign

April 16, 2022

ILLINOIS

Outline

1. Introduction
2. Useful practices
   - Version control
   - Open development workflow
   - Automation and continuous integration
3. Conclusion

About me

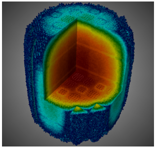
- Advanced Reactors and Fuel Cycles (ARFC) group at UIUC
- Software tools for development, verification, and lisencing of advanced reactors
- Open source

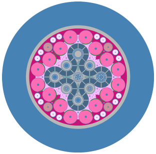
Software

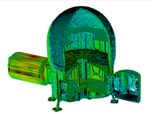
Sources: [3], [7], [17]

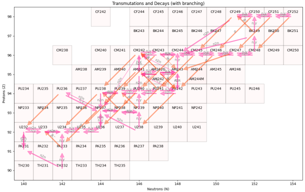
- Ultimately, software is a *tool* we can use to solve (potentially) complex problems.

What kinds of problems do *we* use software to solve?



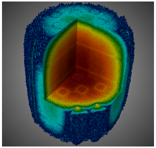


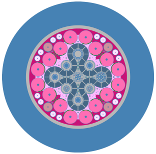


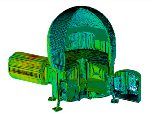


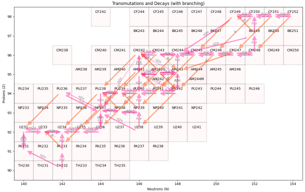
Sources: [19],[14],[13],[9]

What kinds of problems do *we* use software to solve?









Sources: [19],[14],[13],[9]

- Neutron transport
- Thermal hydraulics
- Accident analysis
- Materials
- Decay chains
- PRA

Advanced reactor modeling

Regulatory bodies will require new software features in order to effectively and efficiently perform licensing activities for the next generation of reactor designs[20]

Advanced reactor modeling

Regulatory bodies will require new software features in order to effectively and efficiently perform licensing activities for the next generation of reactor designs[20]

IAEA facilitated ONCORE initiative[12]:

"ONCORE. . . is an IAEA-facilitated international collaboration framework for the development and applictaion of open-source multi-physics simulation tools to support research, education, and training for analysis of advanced nuclear power reactors" [15]

Open source software

Software whose source code is public.

- Promotes collaborative contributions
- Reduces duplicate work

Open source software

Software whose source code is public.

- Promotes collaborative contributions
- Reduces duplicate work

A sample of open-source codes in the nuclear space

- OpenMC (monte carlo neutron transport)
- MOOSE (multiphysics finite element framework)
- nekRS (Spectral element computational fluid dynamics)

How to develop features in open-source software?

There's no "right" way to do this, but there are useful conventions and concepts:

- Code standards (e.g. PEP8, The C Standard)
- User and developer guides
  - Installation instructions
  - API documentation
  - Contributing guidelines
- **Version control**
- **Open development**
- **Automation**

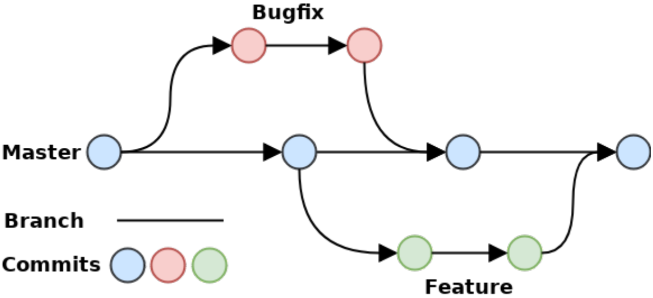
These conventions and practices work in closed codes as well!

Outline

1. Introduction
2. Useful practices
   - Version control
   - Open development workflow
   - Automation and continuous integration
3. Conclusion

Version Control

Version control is the practice of the tracking changes.



Commits retain information about who made them, preserving attribution and authorship wihtout needing to store metadata in source files themselves.

Version Control Systems

Sources: [17], [18], [8]

Software that tracks changes via commits.

Basic workflow:

1. Make changes to tracked files in your local repository
2. Stage the changes.
3. Commit the stages changes to the repository
4. Push the commited changes to a remote repository (where the official version of the code is hosted)

A real-life git example

Fixing a typo in OpenMC

Excerpt of `openmc/surface.py` (lines 1970 - 1972)

```
class ZCone(QuadricMixin, Surface):
    """A cone parallel to the x-axis of the form :math:`(x - x_0
    )^2 + (y - y_0)^2 = r^2 (z - z_0)^2`.
```

Make our fix:

```
class ZCone(QuadricMixin, Surface):
    """A cone parallel to the z-axis of the form :math:`(x - x_0
    )^2 + (y - y_0)^2 = r^2 (z - z_0)^2`.
```

A real-life git example

Fixing a typo in OpenMC

In the shell:

```
user@computer1:~/openmc$ git add openmc/surface.py
user@computer1:~/openmc$ git commit -m "fix axis spec in docstri
ng for ZCone"
user@computer1:~/openmc$ git push
```

In this case, we pushed to the openmc-dev/openmc repository on GitHub. The commit is here → https://github.com/openmc-dev/openmc/pull/2018/commits/48dbf1a4c3a83bf7abd0722ab868f532abc6b5bd

Open development

Open *source*: hosting code publicly

Open *development*: is a set of development practices that emphasizes reproducibility and searchability:

- Verbose commit messages
- Ticketing system to track bugs and feature proposals
- Robust justification for bug fixes and features

Web-based development platforms like GitHub, GitLab, and BitBucket all provide interfaces that can accomodate an open development approach, in additon to hosting open souurce code.

Sources: [5], [6], [1]

Why open development?

Open development *leverages the expertise of the community*, leading to **more robust software**.

Open development decision making process is well documented, simplifying onboarding of new and external developers.

Closed codes can adopt open development practices too!

For more on open development, check out *Working in Public* by Nadia Eghbal [11]

Open development example

Implementing OpenMC in SaltProc

Idea: Implement OpenMC in an open-source Molten Salt Reactor depletion simulator[1]

In the issue tracker, I detail background/motivation and the description of the idea:

Background and motivation

I will be splitting up the implementation of `DepcodeOpenMC` into two parts to reduce PR bloat without sacrificing development of features that have a strong coupling based on implementation decisions.

Description of idea

We need to have functions that can perform reading/writing of input files, that can run OpenMC depletion, and that support the previous two tasks.

[1] You can find the issue here: https://github.com/arfc/saltproc/issues/133

Open development example

Implementing OpenMC in SaltProc

I also detail a skeleton design/implementation:

Implementation details

- Input file reading/writing
  - We'll need to modify the input file structure since OpenMC has input settings split across multiple files. If possible, we should preserve the current input file structure for serpent. It's possible there's a way to do this with JSON Schema.
  - We need functions that store run-time versions of the materials, settings, and geometry files.
  - We need functions that pass saltproc simulation information to openmc materials and settings, as well as the relevant parameters in the `openmc.deplete` module.
- Running the depletion simulation
  - We need a python script that accepts as command line arguments paths to our our OpenMC input files.
  - we will run this script as a sub process in `DepcodeOpenMC.run_depcode` function.
  - we will execute this script from `mpiexec` and pass the relevant parameters
- Tests
  - We need to write unit tests for each new function (where possible. Can't really do this for the `run_depcode` function)
  - A script to convert serpent material and geometry files to the openmc form would be nice to have

Open development example

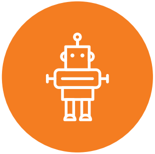
Implementing OpenMC in SaltProc

Finally, I write down any snags I can think of:

Potential snags

- We'll need to modify the input file structure since OpenMC has input settings split across multiple files
- Any API changes means we need to update tests. It's best to do this slowly and one at a time so we can catch and handle errors efficiently.

Automation



Source: [16]

Automating out repetitive tasks saves more time for designing and developing features and bug fixes.

Automation frameworks provide a configurable and tested method to create and execute automated tasks.

Automation Framework

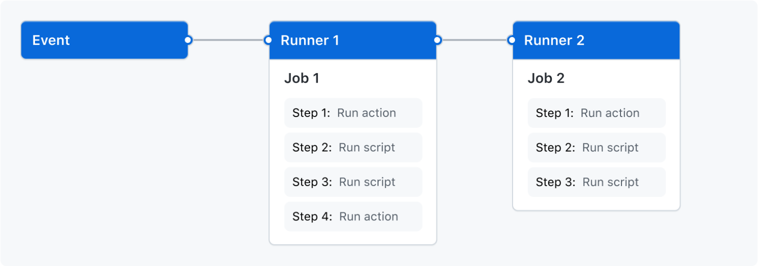
Sources: [2], [4]

Automation frameworks are services execute user-created instruction sets called *workflows* when certain conditions, or *triggers*, are met.
Automation frameworks can read files from the main host repository. This enables users to create workflows to do things like:

- automatically run a test suite whenever the code changes (*continuous integration*)
- build and deploy online documentation
- manage repository metadata

GitHub Actions

GitHub Actions is an automation framework integrated into every GitHub repository.



Source: [10]

The basic workflow file structure is as follows:

- Workflow name
- Define workflow triggering events
- Define the workflow jobs and steps

GitHub Actions Workflow Example

Populating SaltProc release notes

```
# Preamble
name: Populate SaltProc release notes

on:
  push:
    branches:
      - 'master'
    paths:
      - 'doc/releasenotes/v**.rst'
  # enable worflow to be run manually
  workflow_dispatch:

jobs:
  populate-releasenotes:
    runs-on: ubuntu-latest
    defaults:
      run:
        shell: bash -l {0}
```

GitHub Actions Workflow Example

Populating SaltProc release notes

```
steps:
  - uses: actions/checkout@v2

  - name: Set up Python 3.9
    uses: actions/setup-python@v2
    with:
      python-version: 3.9
  - name: Add conda to system path
    run: |
      # $CONDA is an environment variable pointing to the
      # root of the miniconda directory
      echo $CONDA/bin >> $GITHUB_PATH

  - name: install pandoc
    run: |
      conda install -c conda-forge pandoc
      pip install --upgrade pandoc
```

GitHub Actions Workflow Example

Populating SaltProc release notes

```
- name: Get most recent draft release version
  run: |
    echo "RELEASE_VERSION=$(gh api repos/
    ${{ github.repository }}/
    releases --jq '.[0] | .name')" >> $GITHUB_ENV
    echo "RELEASE_ID=$(gh api repos/
    ${{ github.repository }}/
    releases --jq '.[0] | .id')" >> $GITHUB_ENV
  env:
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

- name: Convert .rst to .md
  run: |
    pandoc -o RELEASENOTES.md -f rst -t gfm doc/releasenotes/
    ${{ env.RELEASE_VERSION }}.rst --columns 1000
    sed -i "s/# Release notes for ${{ env.RELEASE_VERSION }}//g"
    RELEASENOTES.md
```

GitHub Actions Workflow Example

Populating SaltProc release notes

```
- name: Populate the release description with RELEASENOTES.md
  run: |
    CURRENT_TAG=$(gh api repos/${{ github.repository }}/
    releases/${{ env.RELEASE_ID }} \
    -H "Authorize: token ${{ secrets.GITHUB_TOKEN }}" \
    -H "Accept: application/vnd.github.v3+json" \
    -X GET \
    --jq '.tag_name')
    gh api repos/${{ github.repository }}/releases/
    ${{ env.RELEASE_ID }} \
    -H "Authorize: token ${{ secrets.GITHUB_TOKEN }}" \
    -H "Accept: application/vnd.github.v3+json" \
    -X PATCH \
    -F tag_name=$CURRENT_TAG \
    -F body="$(cat RELEASENOTES.md)"
  env:
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Outline

1. Introduction
2. Useful practices
   - Version control
   - Open development workflow
   - Automation and continuous integration
3. Conclusion

Main takeaways

- Open source software is becoming an important component of modeling and simulation for advanced reactors.
- Version control helps us keep track of changes and work on multiple features at once.
- Open development makes our code more robust and ensures new and external contributors can understand design and implementation desicions.
- Automating out repetitive tasks helps catch bugs and streamline the development workflow.

Acknowledgement

ARFC group members
NRC Integrated University Grant Program Fellowship

References I

[1] Bitbucket Logo.

[2] CircleCI Logo.

[3] Firefox Logo.

[4] Github Actions Logo.

[5] Github Logo.

[6] GitLab Logo.

[7] OpenMC Logo.

[8] Subversion Logo.

[9] ARMI 0.2.3 documentation.
Transmutation and decay reactions.

[10] GitHub Actions Documentation.
Understanding GitHub actions.

References II

[11] Nadia Eghbal.
Working in public : the making and maintenance of open source software.
Stripe Press, San Francisco, CA, 2020.

[12] C. Fiorina, P. Shriwise, A. Dufresne, J. Ragusa, K. Ivanov, T. Valentine, B. Lindley, S. Kelm, E. Shwageraus, S. Monti, C. Batra, A. Pautz, S. Lorenzi, P. Rubiolo, I. Clifford, and B. Dechenaux.
AN INITIATIVE FOR THE DEVELOPMENT AND APPLICATION OF OPEN-SOURCE MULTI-PHYSICS SIMULATION IN SUPPORT OF r&d AND e&t IN NUCLEAR SCIENCE AND TECHNOLOGY.
247:02040.
Publisher: EDP Sciences.

[13] CNERG group website.
DAGMC: Direct Accelerated Geometry Monte Carlo.

[14] OpenMC User's Guide.
Chapter 9. Geometry Visualization.

[15] IAEA.
Open-source nuclear codes for reactor analysis - home.

References III

[16] Iyi Kon.
Robot vector icon.

[17] Jason Long.
Git Logo.

[18] Matt Mackall and Cali Mastny.
Mercurial Logo.

[19] ORNL.
ExaSMR: Coupled Monte Carlo Neutronics and Fluid Flow Simulation of Small Modular Reactors.

[20] U.S. Nuclear Regulatory Comission.
NRC non-light water reactor vision and strategy, volume 1 - computer code suite for non-LWR plant systems analysis.