A project-based course on software development for (engineering) research

Kyle Niemeyer
Oregon State University

13 June 2019

kyle.niemeyer@oregonstate.edu
@kyleniemeyer
git.io/nrg

Motivation

The stats:

- 2009: **91%** scientists consider research software important/very important[1]
- UK academics in 2014: **90%** use software in research; **70%** of them would find research impractical without it[2]
- US postdocs in 2017: **95%** / **63%** same[3]

But: practically no graduate students / researchers receive formal training in this area.

(Contrast with typical experimental methods courses)

[1] JE Hannay, HP Langtangen, et al. SECSE 2009. Vancouver, BC, 1–8. https://doi.org/10.1109/SECSE.2009.5069155
[2] S Hettrick et al. *UK Research Software Survey 2014*. (2015) https://doi.org/10.7488/ds/253
[3] U Nangia & DS Katz, WSSSPE 5.1 (2017) https://doi.org/10.5281/zenodo.814102

Computational Science?

Computational Science?

software / computational skills : computational science

Computational Science?

software / computational skills : computational science

::

Computational Science?

software / computational skills : computational science

::

experimental skills : experimental science

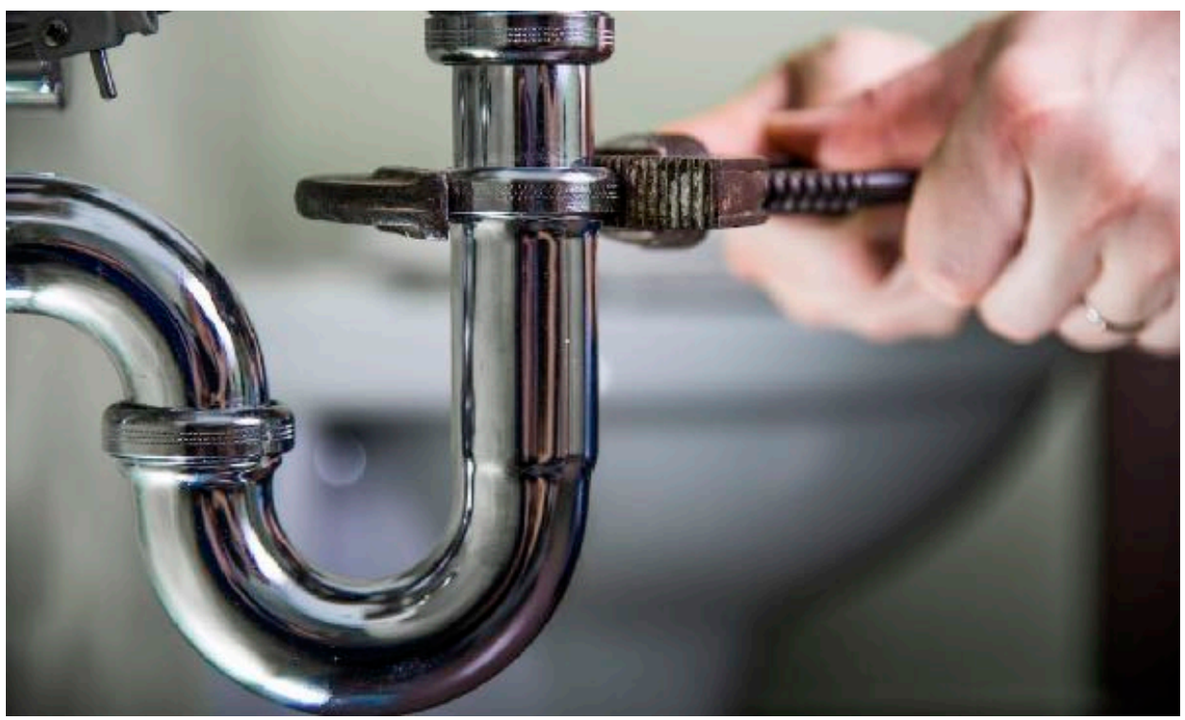
https://edition.cnn.com/travel/article/spectacular-fountains/index.html

https://edition.cnn.com/travel/article/spectacular-fountains/index.html

needs

https://edition.cnn.com/travel/article/spectacular-fountains/index.html

needs



http://artisanplumbing.ca

Who am I?

- Assistant Professor of Mechanical Engineering, Oregon State University

- Research focuses on simulations and modeling of reacting fluid flows and chemical kinetics

- Associate Editor-in-Chief, *Journal of Open Source Software*

at NCM 2019 in Pasadena!

Kyle Niemeyer

kyleniemeyer

Developer Program Member

PRO

Edit profile

Assistant Professor of mechanical engineering. Leads **@Niemeyer-Research-Group**

Oregon State University

Corvallis, OR

kyle.niemeyer@gmail.com

http://kyleniemeyer.com

Organizations

Overview | Repositories 87 | Projects 0 | Stars 192 | Followers 61 | Following 16

Pinned

Customize your pins

SLACKHA/pyJac

Creates C and CUDA analytical Jacobians for chemical kinetics ODE systems

Python ★ 16 15

pr-omethe-us/PyTeCK

Automatically test chemical kinetic models using experimental data

Python ★ 5 6

irrev_mech

Python utility that converts Chemkin-format chemical reaction mechanism with reversible reactions into one with only irreversible reactions

Python ★ 6 3

WSSSPE/meetings

Products from Working towards Sustainable Software for Science: Practice and Experiences (WSSSPE) activities

TeX ★ 16 35

pr-omethe-us/PyKED

Python interface to the ChemKED database format

Python ★ 8 12

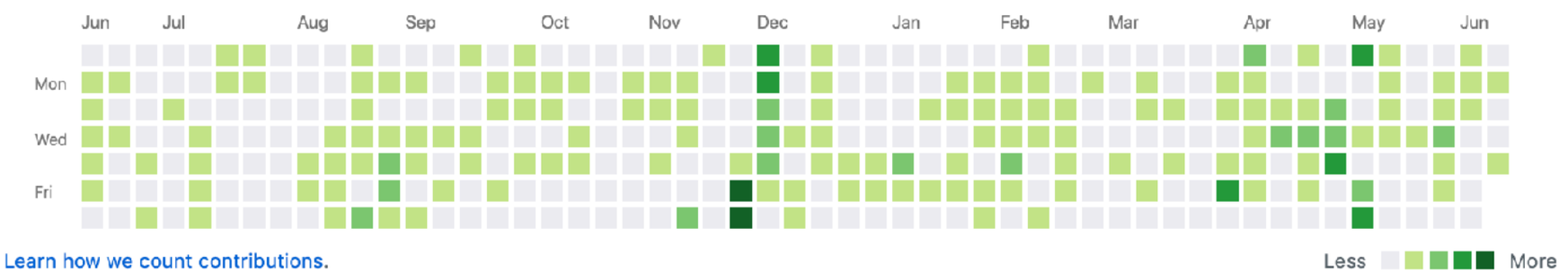
Niemeyer-Research-Group/pyMARS

Python-based (chemical kinetic) Model Automatic Reduction Software

Python ★ 12 16

816 contributions in the last year

Contribution settings



Learn how we count contributions.

Less More

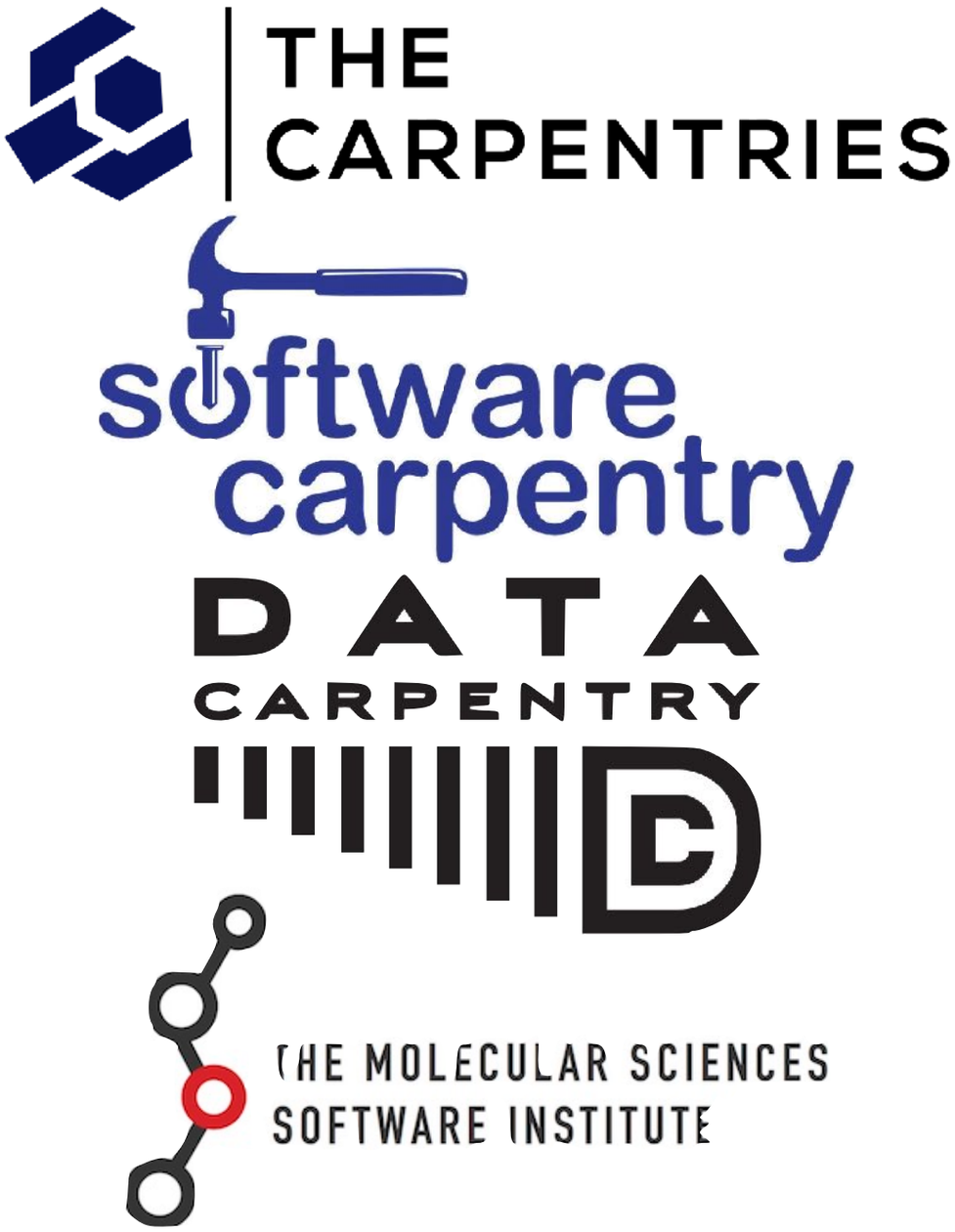
https://github.com/kyleniemeyer

Existing Resources

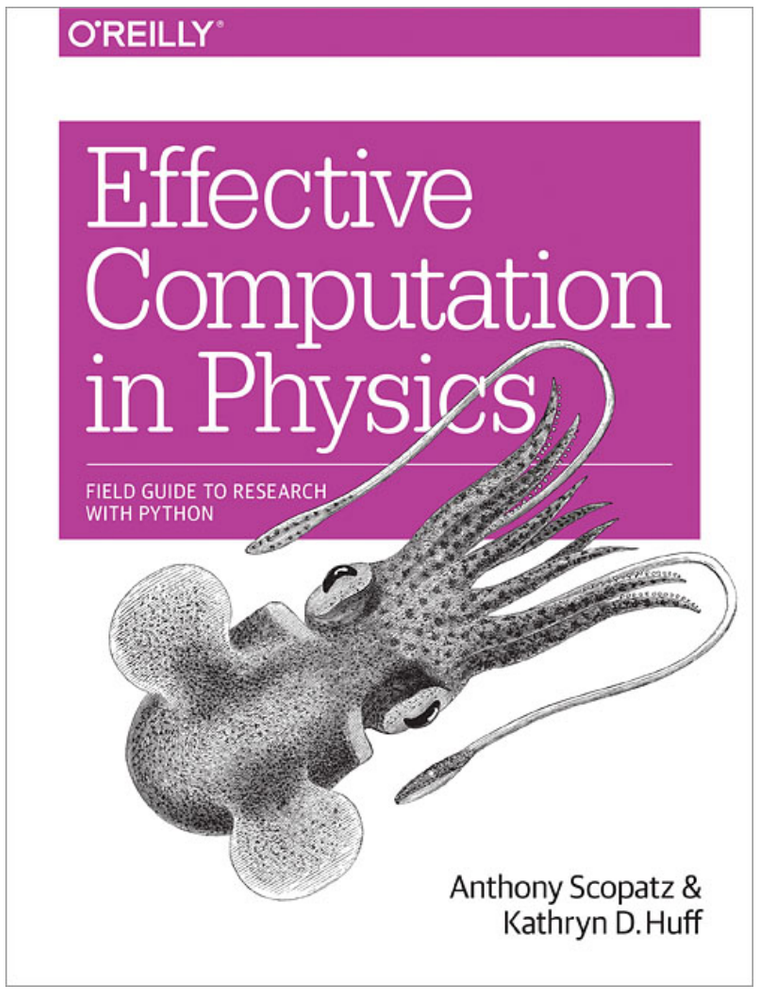
- Software Carpentry workshops have become a popular way to teach fundamental skills: Unix/ command line, version control with Git, and Python programming

- Similarly, Data Carpentry workshops for data science

- For domain scientists: MolSSI Software Summer School

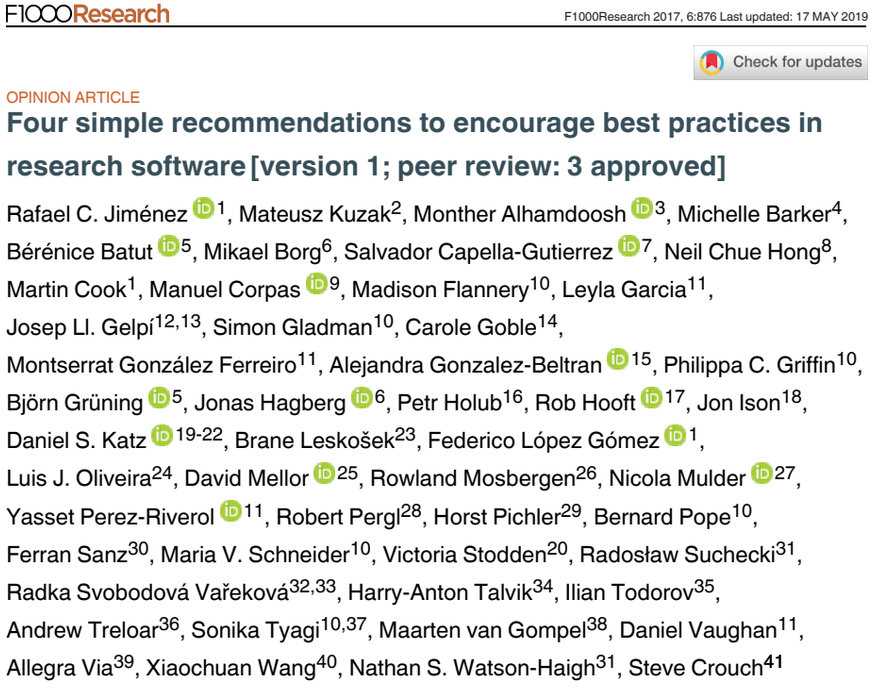


More (Static) Resources



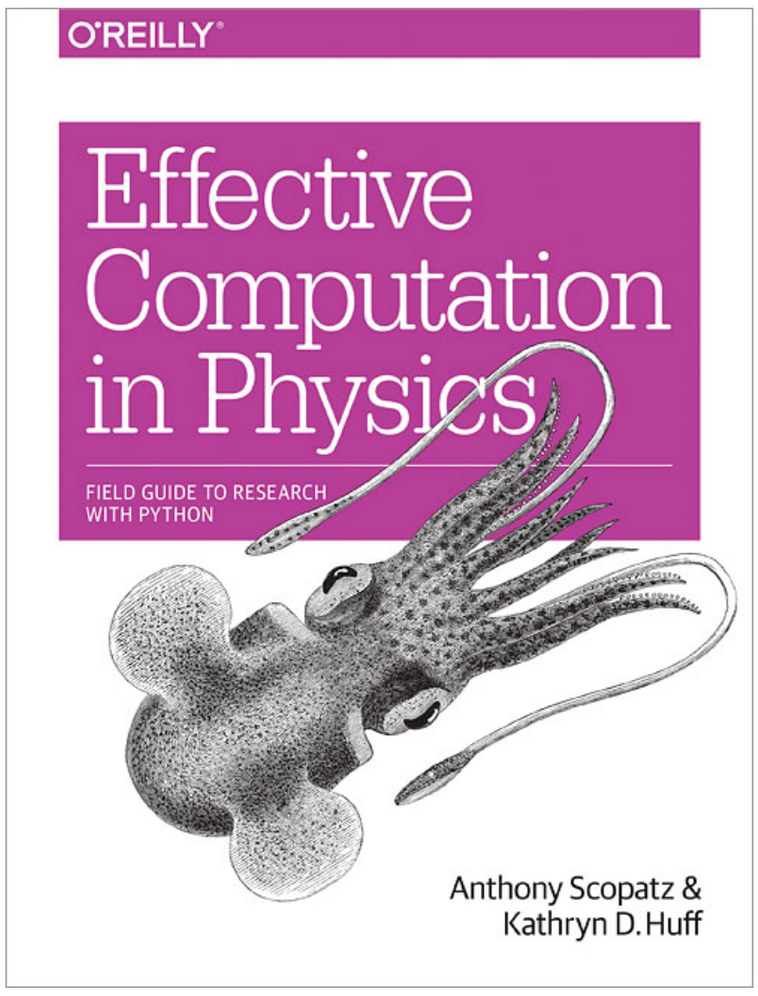
http://physics.codes

https://doi.org/10.1371/ journal.pcbi.1005510



https://doi.org/10.12688/f1000research.11407.1

More (Static) Resources



http://physics.codes

PLOS COMPUTATIONAL BIOLOGY

PERSPECTIVE

Good enough practices in scientific computing

Greg Wilson[1☯*], Jennifer Bryan[2☯], Karen Cranston[3☯], Justin Kitzes[4☯], Lex Nederbragt[5☯], Tracy K. Teal[6☯]

https://doi.org/10.1371/ journal.pcbi.1005510

F1000Research F1000Research 2017, 6:876 Last updated: 17 MAY 2019

Check for updates

OPINION ARTICLE

Four simple recommendations to encourage best practices in research software [version 1; peer review: 3 approved]

Rafael C. Jiménez [1], Mateusz Kuzak[2], Monther Alhamdoosh [3], Michelle Barker[4], Bérénice Batut [5], Mikael Borg[6], Salvador Capella-Gutierrez [7], Neil Chue Hong[8], Martin Cook[1], Manuel Corpas [9], Madison Flannery[10], Leyla Garcia[11], Josep Ll. Gelpí[12,13], Simon Gladman[10], Carole Goble[14], Montserrat González Ferreiro[11], Alejandra Gonzalez-Beltran [15], Philippa C. Griffin[10], Björn Grüning [5], Jonas Hagberg [6], Petr Holub[16], Rob Hooft [17], Jon Ison[18], Daniel S. Katz [19-22], Brane Leskošek[23], Federico López Gómez [1], Luis J. Oliveira[24], David Mellor [25], Rowland Mosbergen[26], Nicola Mulder [27], Yasset Perez-Riverol [11], Robert Pergl[28], Horst Pichler[29], Bernard Pope[10], Ferran Sanz[30], Maria V. Schneider[10], Victoria Stodden[20], Radosław Suchecki[31], Radka Svobodová Vařeková[32,33], Harry-Anton Talvik[34], Ilian Todorov[35], Andrew Treloar[36], Sonika Tyagi[10,37], Maarten van Gompel[38], Daniel Vaughan[11], Allegra Via[39], Xiaochuan Wang[40], Nathan S. Watson-Haigh[31], Steve Crouch[41]

https://doi.org/10.12688/f1000research.11407.1

Big influence on my class!

My Solution:

A 10-week (primarily) graduate course to fill this gap, using existing resources as inspiration/building blocks.

Course Philosophy

Course Philosophy

- **Main goal:** teach students best practices and practical skills to develop useful, tested, and (potentially) sustainable research software.

Course Philosophy

- **Main goal:** teach students best practices and practical skills to develop useful, tested, and (potentially) sustainable research software.

- Rather than give standalone assignments, all assigned work relates to a quarter-long project, which is intended to support their thesis research.

Course Philosophy

- **Main goal:** teach students best practices and practical skills to develop useful, tested, and (potentially) sustainable research software.

- Rather than give standalone assignments, all assigned work relates to a quarter-long project, which is intended to support their thesis research.

- Also try to instill open science as the default—relatively early in students' careers.

Course Philosophy

- **Main goal:** teach students best practices and practical skills to develop useful, tested, and (potentially) sustainable research software.

- Rather than give standalone assignments, all assigned work relates to a quarter-long project, which is intended to support their thesis research.

- Also try to instill open science as the default—relatively early in students' careers.

- Classes mix a typical presentation-based lecture approach with interactive/hands-on work and discussion.

Topics Taught

- Version control with Git
- Remote and collaborative version control with GitHub
- Licensing and copyright
- Testing software, test coverage, and continuous integration
- Structuring Python packages
- Writing documentation
- Objects and classes in Python
- Working with files
- Building command-line interfaces
- Packaging and distributing your Python software
- Introduction to parallel programming
- Open science, software citation, and reproducibility best practices
- Documentation with Sphinx

Example:

module on testing

https://softwaredevengresearch.github.io/lecture-packaging-testing/

Testing your code

How do you know your code gives the right answers?

... what about after you make changes?

Legacy code: ”code without tests” (Michael Feathers, *Working Effectively with Legacy Code*)

Testing your code

when to test: **always.**

where to test: **external test suite.**

Example: `tests` subdirectory inside package.

Perfect is the enemy of good; a basic level of tests is better than nothing. But a rigorous test suite will save you time and potential problems in the long run.

Why test?

Testing is a core principle of scientific software; it ensures results are trustworthy.

Scientific and engineering software is used for planes, power plants, satellites, and decisionmaking. Thus, correctness of this software is pretty important.

And we all know how easy it is to have mistakes in code without realizing it...

What and how to test?

```
def kepler_loc(p1, p2, dt, t):
    '''Use Kepler's Laws to predict location of celestial body'''
    ...
    return p3
```

```
def test_kepler_loc():
    p1 = jupiter(two_days_ago)
    p2 = jupiter(yesterday)
    exp = jupiter(today)
    obs = kepler_loc(p1, p2, 1, 1)
    if exp != obs:
        raise ValueError("Jupiter is not where it should be!")
```

What is a test?

Tests compare expected outputs versus observed outputs for known inputs. They do not inspect the body of the function directly. In fact, the body of a function does not even have to exist for a valid test to be written.

```
def test_func():
    exp = get_expected()
    obs = func(*args, **kwargs)
    assert exp == obs
```

Good idea: test through assertions

For exactness:

```
def test_kepler_loc():
    p1 = jupiter(two_days_ago)
    p2 = jupiter(yesterday)
    exp = jupiter(today)
    obs = kepler_loc(p1, p2, 1, 1)
    assert exp == obs
```

For approximate exactness:

```
import numpy as np
def test_kepler_loc():
    p1 = jupiter(two_days_ago)
    p2 = jupiter(yesterday)
    exp = jupiter(today)
    obs = kepler_loc(p1, p2, 1, 1)
    assert np.allclose(exp, obs)
```

Test using `pytest`

```
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```

```
$ pytest
```

`pytest` finds all testing modules and functions, and runs them.

Kinds of tests

interior test: precise points/values do not matter

edge test: test examines beginning or end of a range

Best practice: test all edges and at least one interior point.

Also **corner cases**: two or more edge cases combined.

```
import numpy as np

def sinc2d(x, y):
    '''(Describe the function here)'''
    if x == 0.0 and y == 0.0:
        return 1.0
    elif x == 0.0:
        return np.sin(y) / y
    elif y == 0.0:
        return np.sin(x) / x
    else:
        return (np.sin(x) / x) * (np.sin(y) / y)
```

```
import numpy as np
from mod import sinc2d

def def test_internal():
    exp = (2.0 / np.pi) * (-2.0 / (3.0 * np.pi))
    obs = sinc2d(np.pi / 2.0, 3.0 * np.pi / 2.0)
    assert np.allclose(exp, obs)
def test_edge_x():
    exp = (-2.0 / (3.0 * np.pi))
    obs = sinc2d(0.0, 3.0 * np.pi / 2.0)
    assert np.allclose(exp, obs)
```

```
def test_edge_y():
    exp = (2.0 / np.pi)
    obs = sinc2d(np.pi / 2.0, 0.0)
    assert np.allclose(exp, obs)
def test_corner():
    exp = 1.0
    obs = sinc2d(0.0, 0.0)
    assert np.allclose(exp, obs)
```

Types of tests

- **unit test**: interrogate individual functions and methods
- **integration test**: verify that multiple pieces of the code work together
- **regression test**: confirm that results match prior code results (which are assumed correct)

Test generators

```
import numpy as np
import pytest

# contents of add.py
def add2(x, y):
    return x + y

class Test(object):
@pytest.mark.parametrize('exp, x, y', [
    (4, 2, 2),
    (5, -5, 10),
    (42, 40, 2),
    (16, 3, 13),
    (-128, 0, -128),
])
def test_add2(self, x, y, exp):
```

```
$ pytest
```

Test-Driven Development (TDD)

Write the tests first.

Before you write any lines of a function, first write the test for that function.

```
from numpy import allclose
from mod import std

def test_std1():
    obs = std([0.0, 2.0])
    exp = 1.0
    assert np.allclose(exp, obs)
```

```
def std(vals):
    # this must be cheating.
    return 1.0
```

```
def test_std1():
    obs = std([0.0, 2.0])
    exp = 1.0
    assert np.allclose(exp, obs)

def test_std2():
    obs = std()
    exp = 0.0
    assert np.allclose(exp, obs)

def test_std3():
    obs = std([0.0, 4.0])
    exp = 2.0
    assert np.allclose(exp, obs)
```

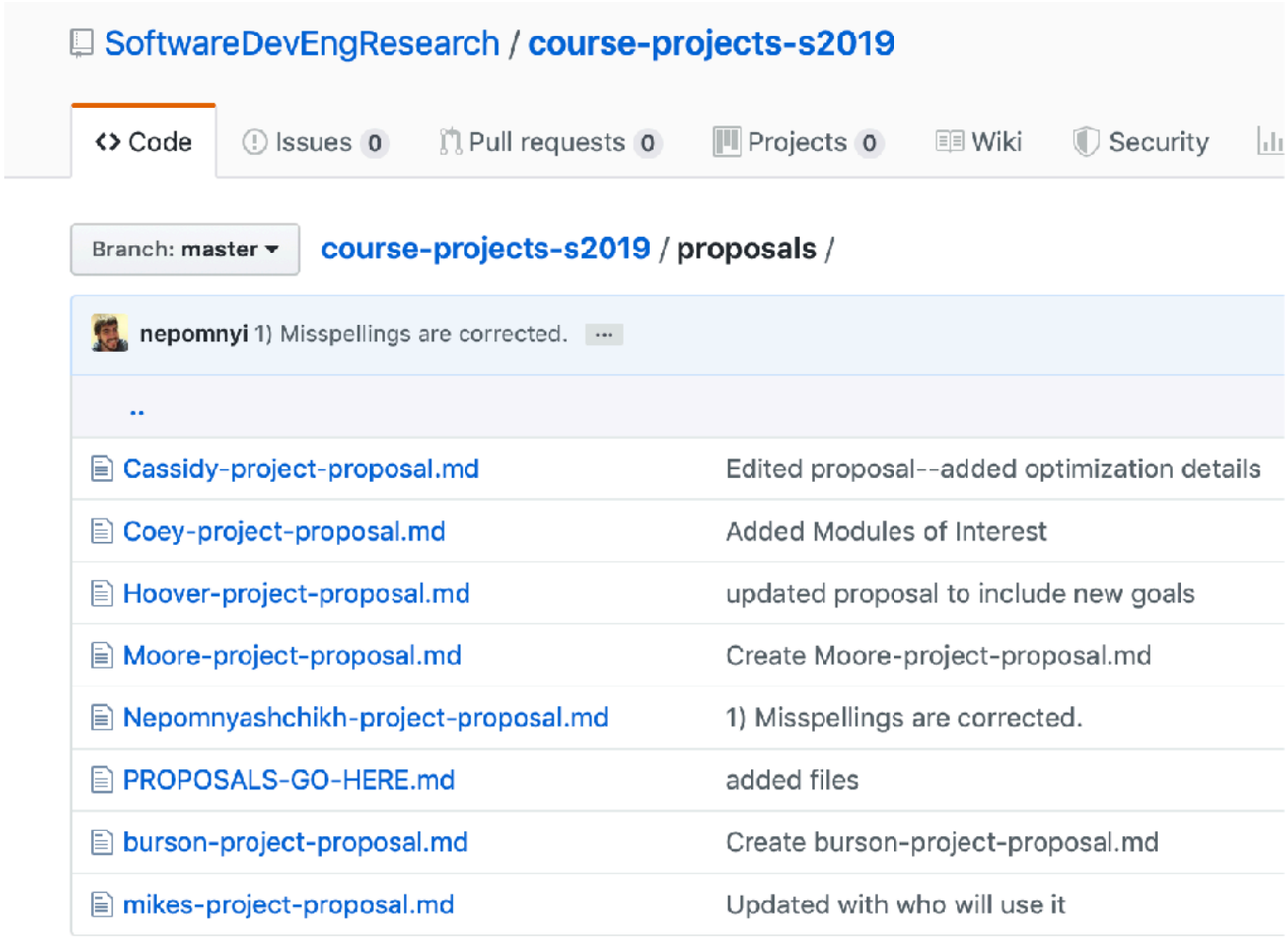
```
def std(vals):
    # a bit better, but still not quite generic
    if len(vals) == 0:
        return 0.0
    return vals[-1] / 2.0
```

```
def test_std1():
    obs = std([0.0, 2.0])
    exp = 1.0
    assert np.allclose(exp, obs)

def test_std2():
    obs = std()
    exp = 0.0
    assert np.allclose(exp, obs)

def test_std3():
    obs = std([0.0, 4.0])
    exp = 2.0
    assert np.allclose(exp, obs)

def test_std4():
    obs = std([1.0, 3.0])
    exp = 1.0
    assert np.allclose(exp, obs)

def test_std4():
    obs = std([1.0, 1.0, 1.0])
    exp = 0.0
    assert np.allclose(exp, obs)
```

```
def std(vals):
    # finally some math
    n = len(vals)
    if n == 0:
        reutnr 0.0
    mu = sum(vals) / n
    var = 0.0
    for val in vals:
        var = var + (val - mu)**2
    return (var / n)**0.5
```

Project

- At beginning of class, students propose a project for the term connected to their thesis research.
- This is submitted as a PR to the course_projects repo



Project Stages

Project Stages

Project proposal

Project Stages

Project proposal

Create project repo & fork to account

Project Stages

Project proposal

Create project repo & fork to account

Create first module with tests, submit as PR for review

Project Stages

- Project proposal
  - Create project repo & fork to account
    - Create first module with tests, submit as PR for review
    - Configure continuous integration

Project Stages

Project proposal

Create project repo & fork to account

Create first module with tests, submit as PR for review

Configure continuous integration

Create command-line interface

Project Stages

Project proposal

Create project repo & fork to account

Create first module with tests, submit as PR for review

Configure continuous integration

Create command-line interface

Package software for distribution

Project Stages

Project proposal

Create project repo & fork to account

Create first module with tests, submit as PR for review

Configure continuous integration

Create command-line interface

Package software for distribution

Configure Zenodo integration

Project Stages

Project proposal

Create project repo & fork to account

Create first module with tests, submit as PR for review

Configure continuous integration

Create command-line interface

Package software for distribution

Configure Zenodo integration

Write report and cite software

Results from 2018–2019

- Offered in Spring 2018 and Spring 2019 as “ME 599: Software Development for Engineering Research”.

- (However, nothing specifically Mechanical Engineering—or even Engineering—about it.)

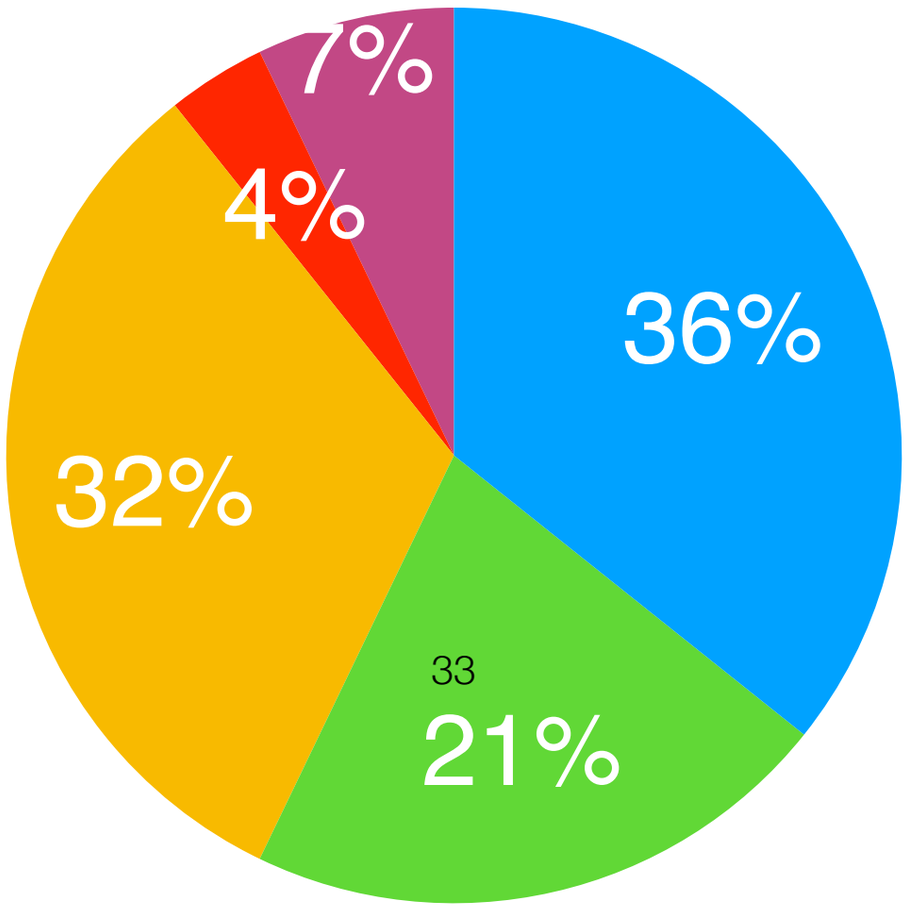
- Enrollment: 17 students in 2018, 9 students in 2019.

Project Topics

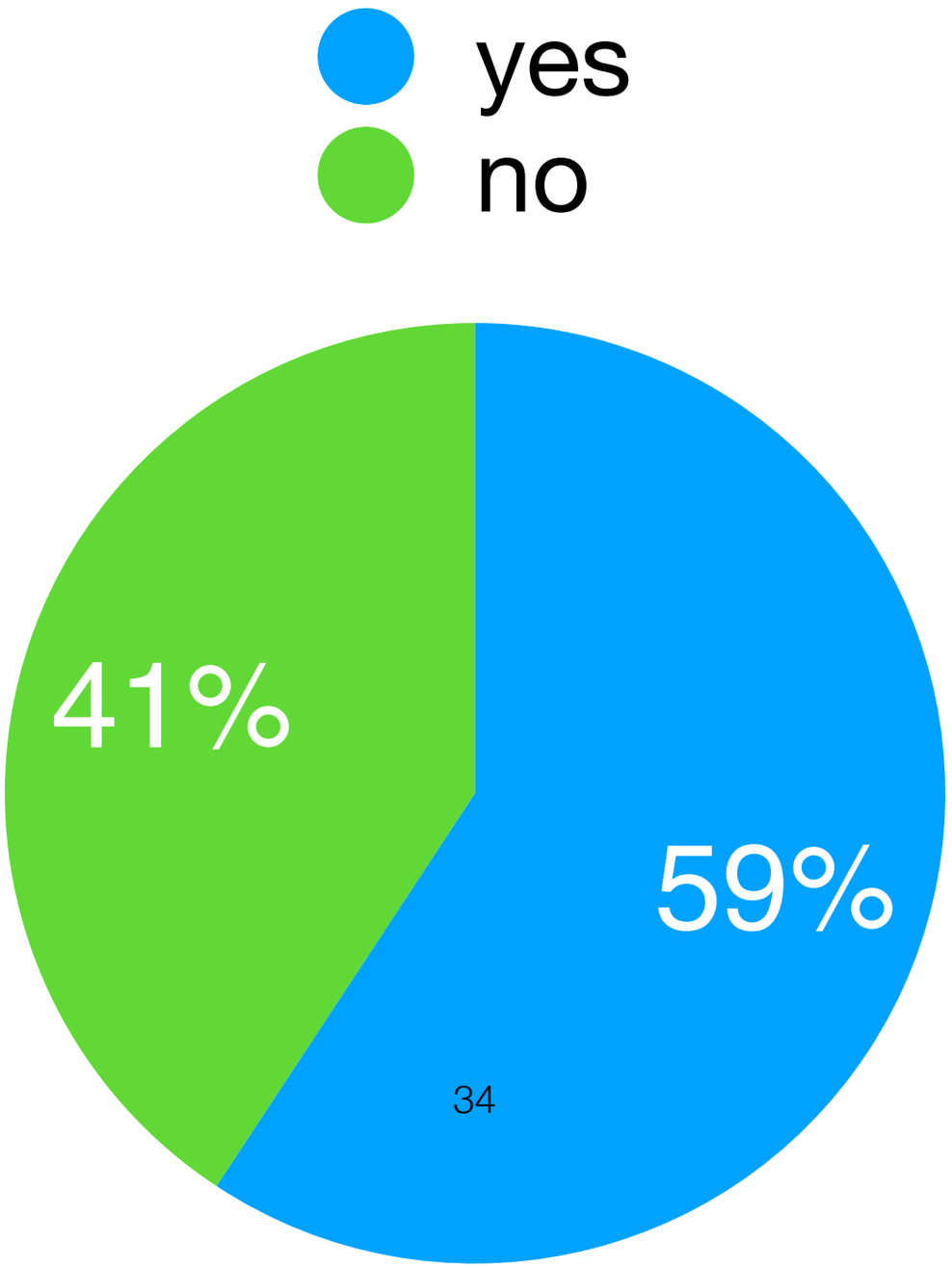
- designing detonation tubes
- extracting features with machine learning from nuclear physics simulations
- interfacing with an 8-channel digital pulse processor board
- simulating and analyzing a formula SAE vehicle engine
- optimizing and analyzing wind-farm layouts
- analyzing spin stabilization of solid rocket motors
- nodal quasi-diffusion solver for nuclear fission
- agent-learning for autonomous path finding
- generating input for Monte-Carlo radiation transport
- calculating solar-energy terms based on location
- analyzing radioxenon spectra
- calculating deep-learning layers for multi-agent reinforcement learners
- analyzing solvent extraction kinetics
- simulating rapid compression machine experiments
- calibrating blackbody infrared cameras
- simulating transient heat transfer in a microchannel
- biomass cookstove optimization tool
- projectile testing prediction
- transit system anomaly detection
- Automatic functional design representation
- Heat exchanger analysis
- Influx modeling for multiphase flows
- Control package for liquid rocket engine test stand

Student composition

- 1st-year MS/PhD student
- 2nd-year MS/PhD student
- ≥3rd-year PhD student
- Undergraduate
- MEng

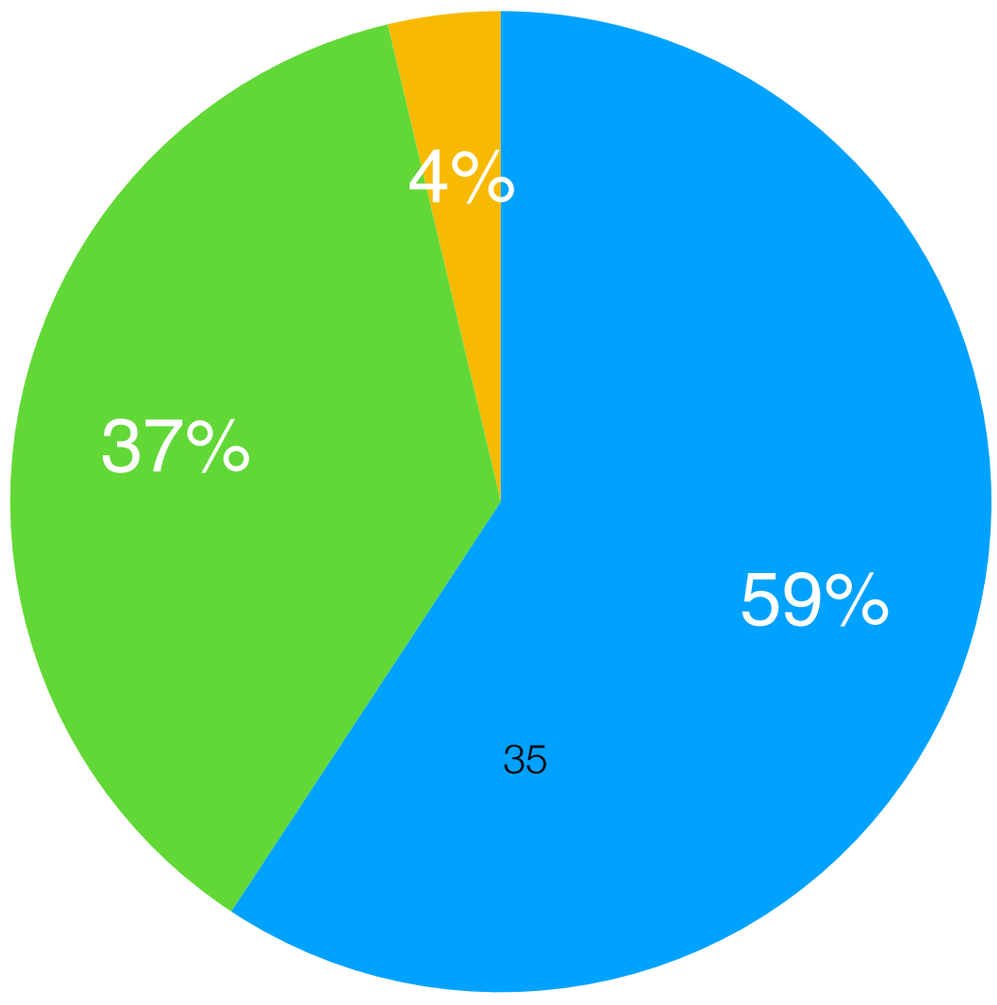


Whether students took Python class



Comfort with command line

- I've used it, though I'm not exactly comfortable with it.
- I'm comfortable working on the command line, but not an expert
- What is the command line?
- I'm a command-line ninja 🤘



Student Feedback

Student Feedback

- In both cases, students rate the course as a whole highly

Student Feedback

- In both cases, students rate the course as a whole highly
- Comments suggest that all graduate students working with software/programming need this material

Student Feedback

- In both cases, students rate the course as a whole highly

- Comments suggest that all graduate students working with software/programming need this material

- Some feelings of being too advanced for their experience level

Post-Course Work

Post-Course Work

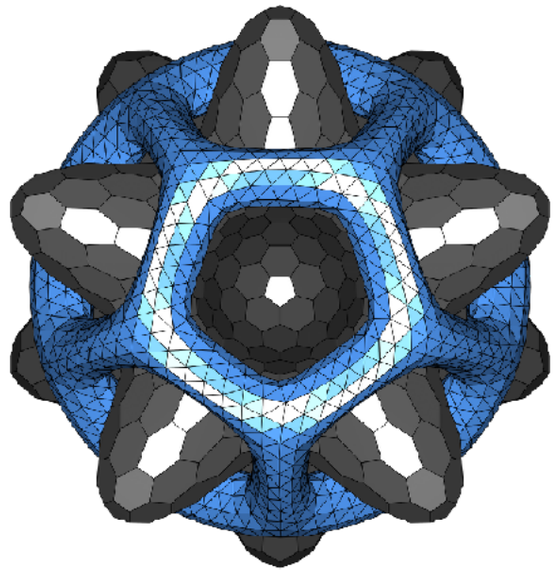
- In both classes so far, roughly half the students indicated that they plan to use and/or develop their packages further for their research

Post-Course Work

- In both classes so far, roughly half the students indicated that they plan to use and/or develop their packages further for their research

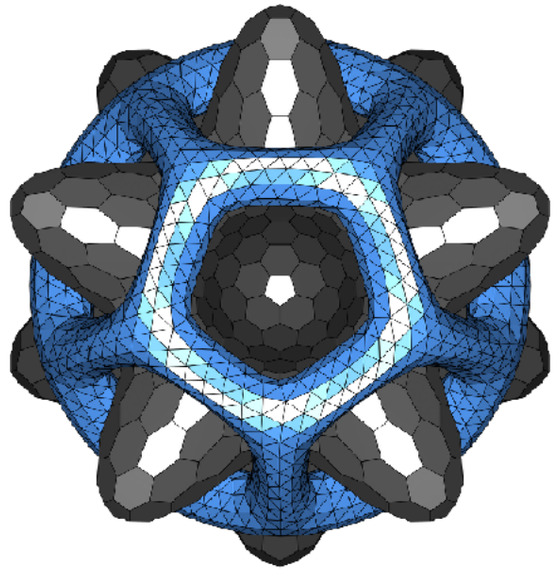
- One ultimate goal: encourage submission to JOSS—one under review, and another to be submitted!

JOSS: Journal of Open Source Software



JOSS: Journal of Open Source Software

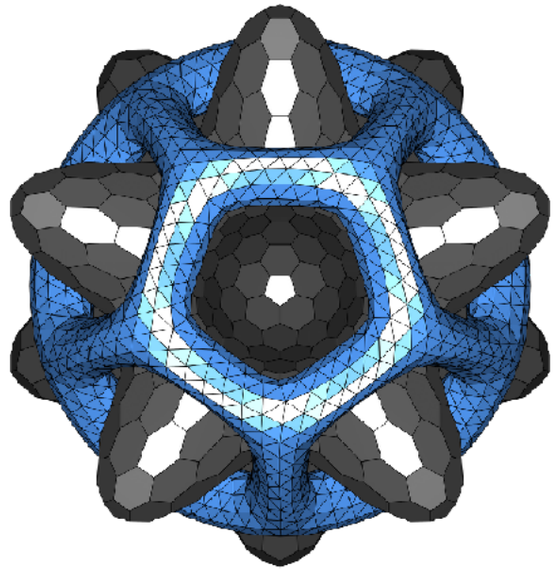
- http://joss.theoj.org/



open source initiative

JOSS: Journal of Open Source Software

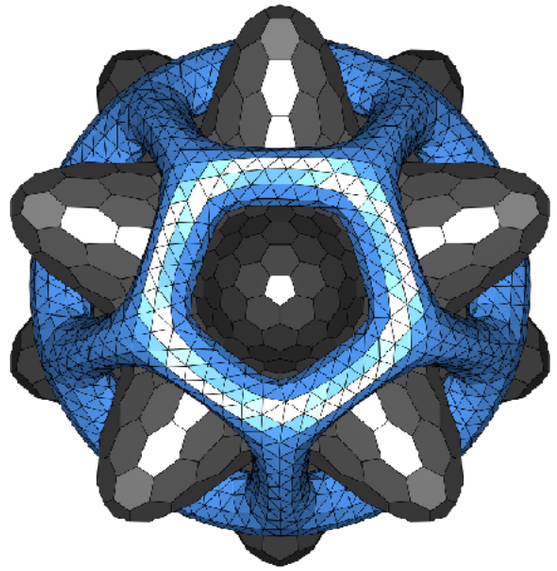
- http://joss.theoj.org/
- Developer-friendly journal for research software packages



open source initiative

JOSS: Journal of Open Source Software

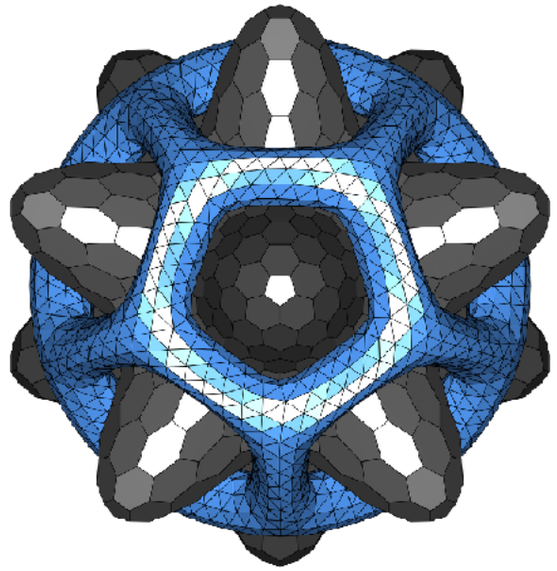
- http://joss.theoj.org/
- Developer-friendly journal for research software packages
- Affiliate of Open Source Initiative



open source initiative

JOSS: Journal of Open Source Software

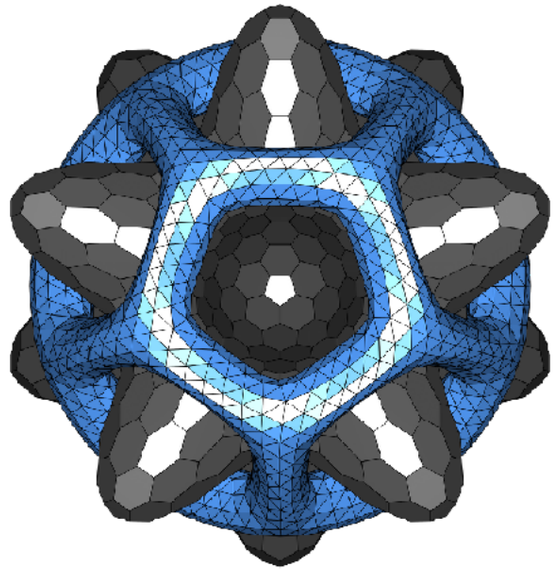
- http://joss.theoj.org/
- Developer-friendly journal for research software packages
- Affiliate of Open Source Initiative
- Open access, no fees



open source initiative

JOSS: Journal of Open Source Software

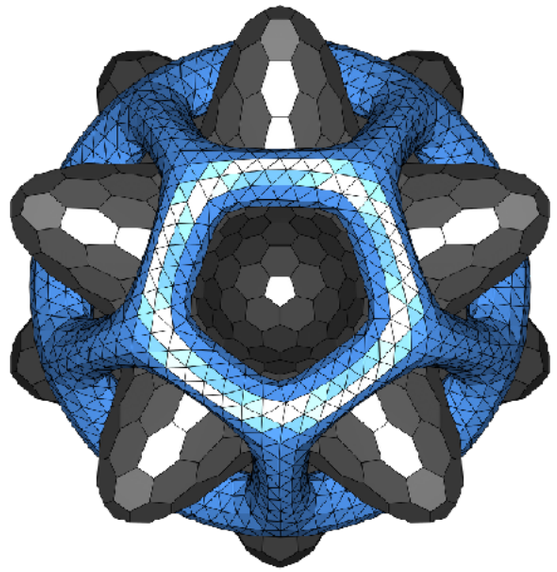
- http://joss.theoj.org/
- Developer-friendly journal for research software packages
- Affiliate of Open Source Initiative
- Open access, no fees
- As of this morning, 597 submissions published



open source initiative

JOSS: Journal of Open Source Software

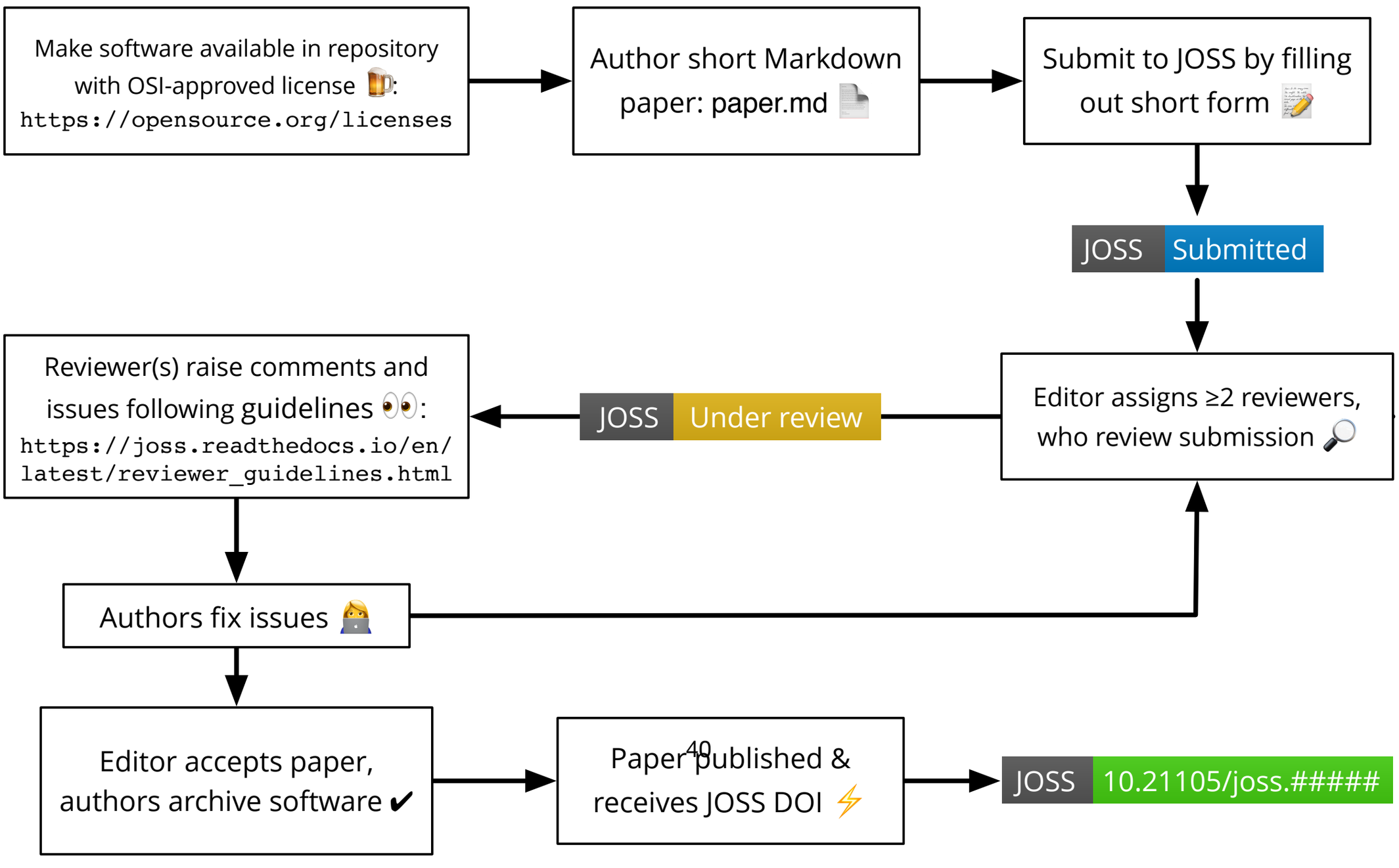
- http://joss.theoj.org/
- Developer-friendly journal for research software packages
- Affiliate of Open Source Initiative
- Open access, no fees
- As of this morning, 597 submissions published

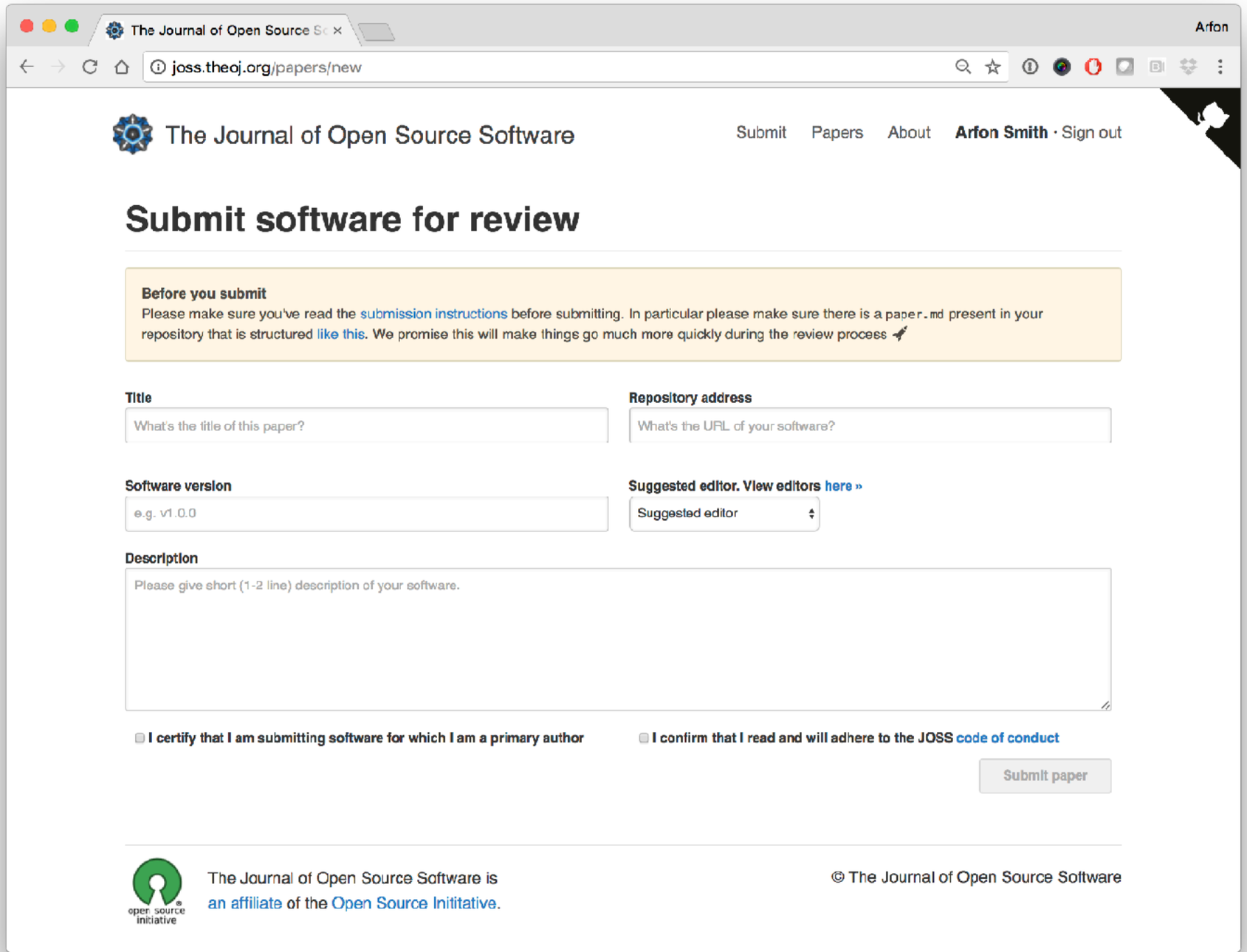


"If you've already licensed your code and have good documentation then we expect that it should take less than an hour to prepare and submit your paper to JOSS."

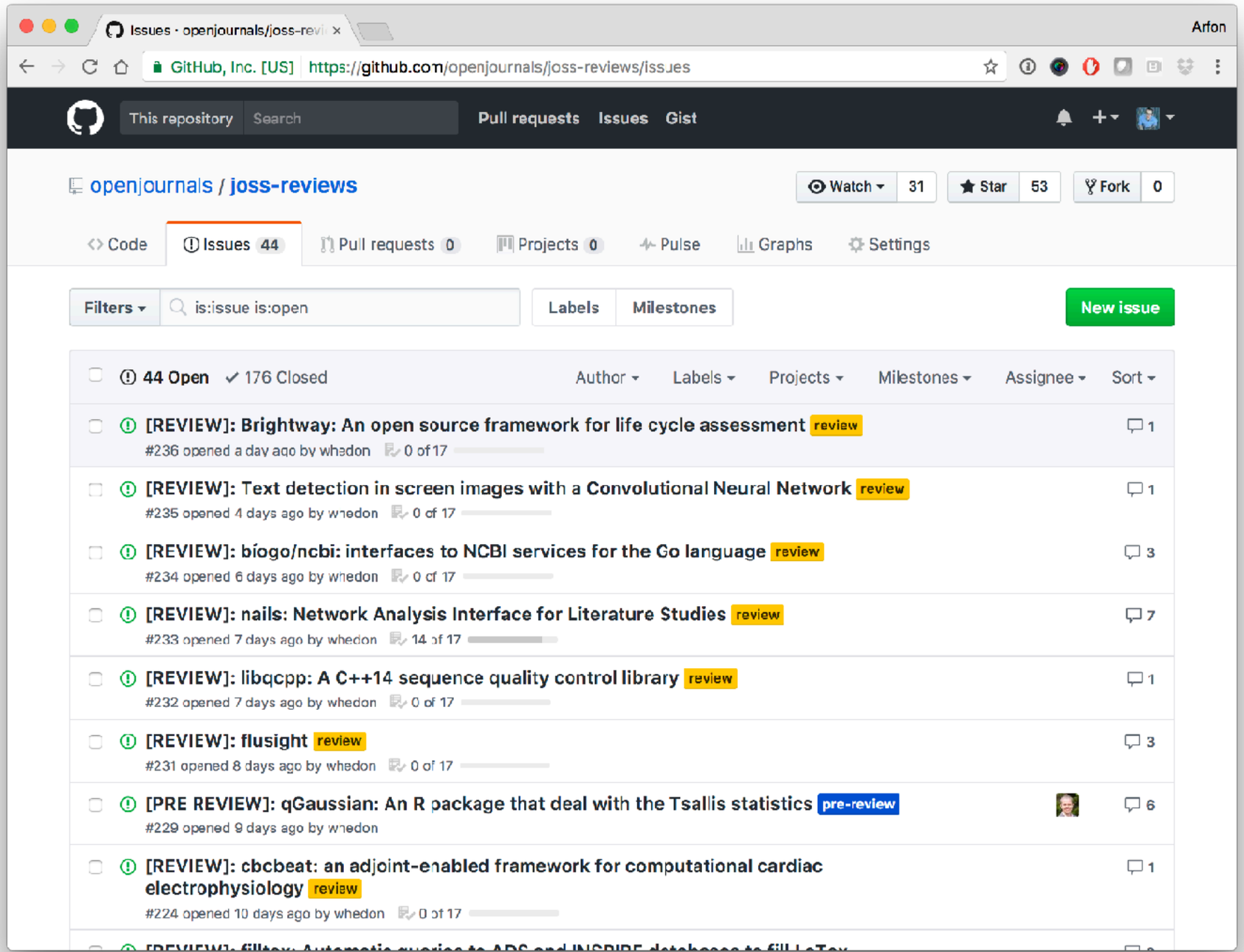
JOSS Editorial Board

JOSS Workflow

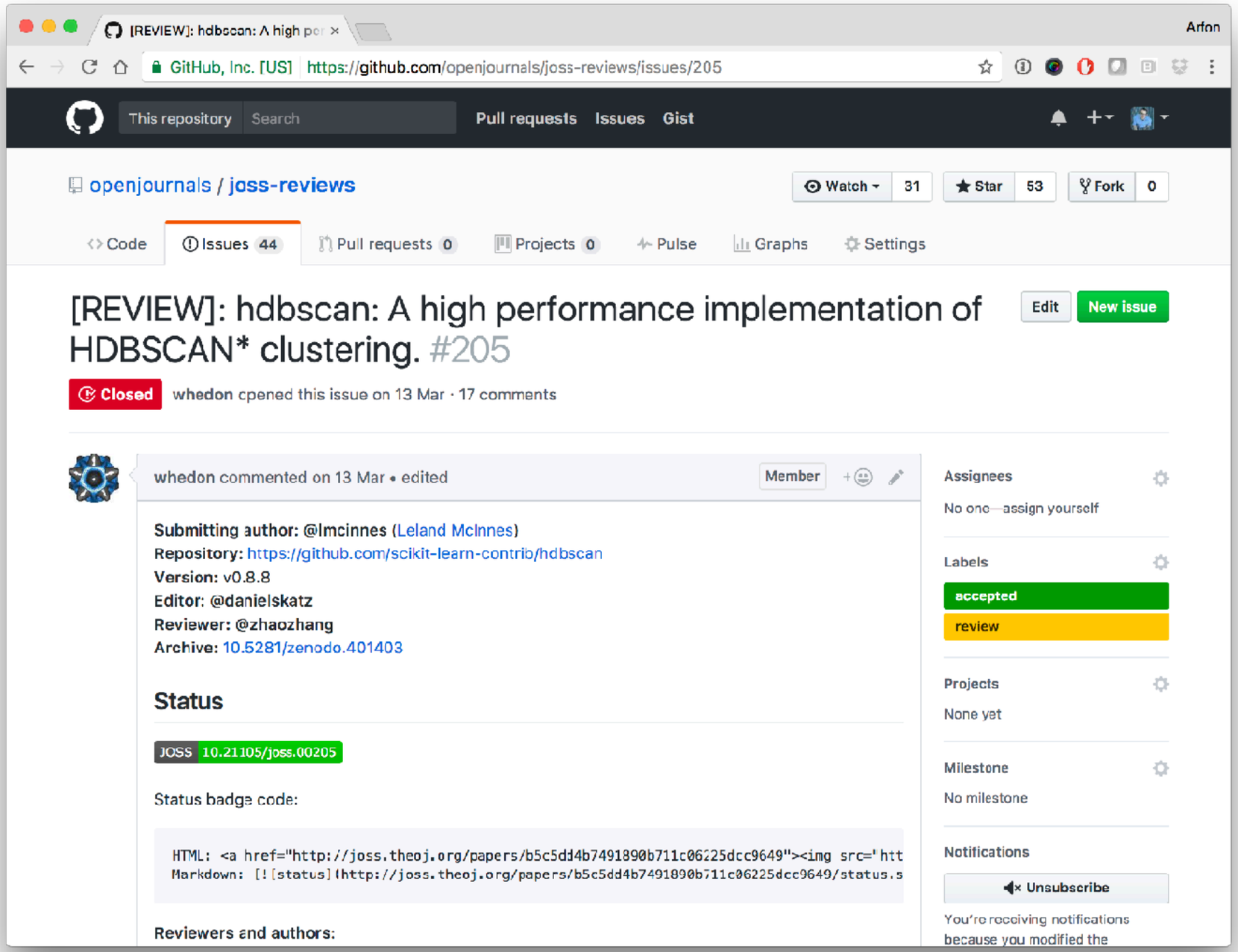




JOSS paper submission

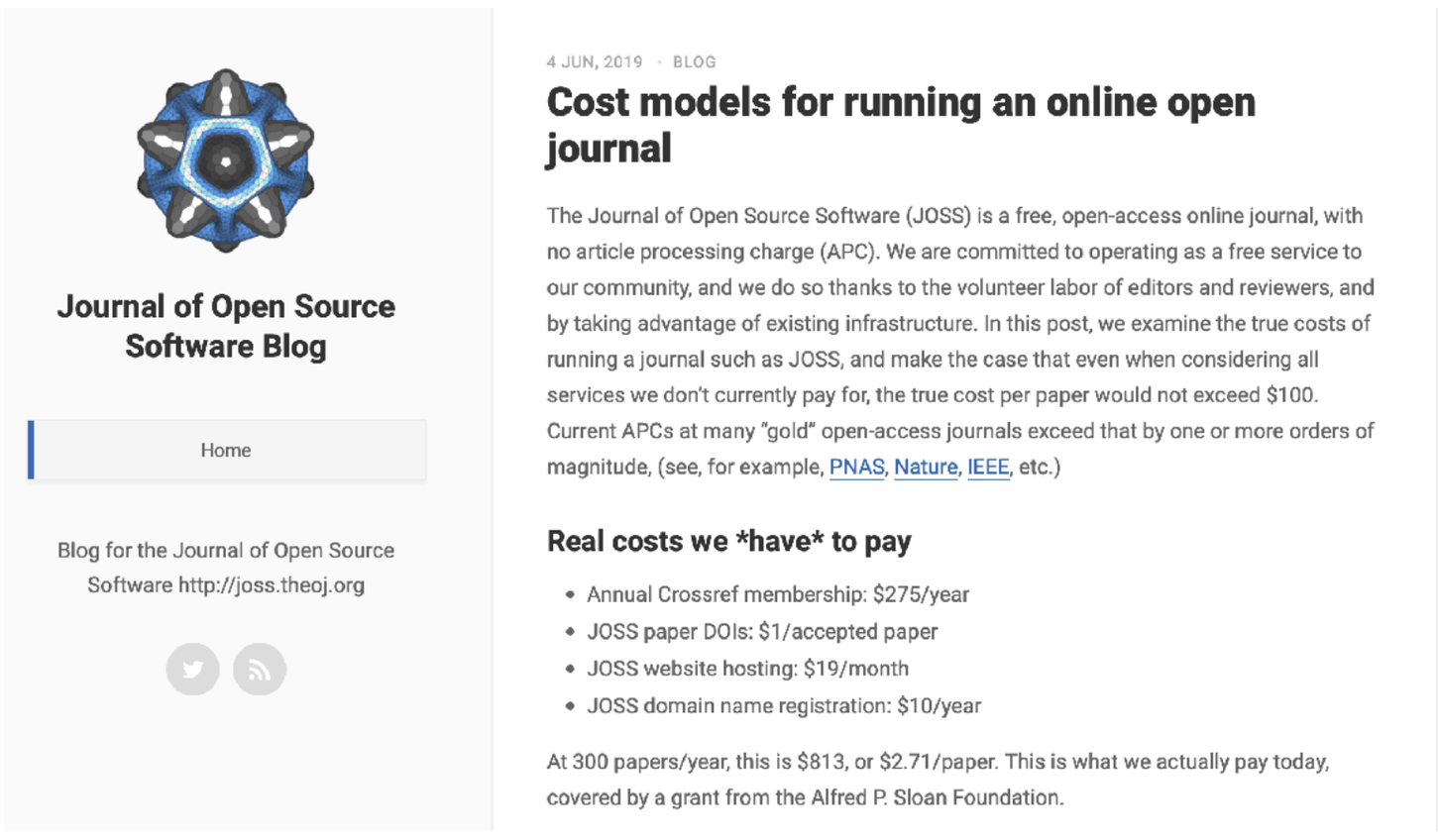


JOSS paper reviews



JOSS paper review

More about JOSS

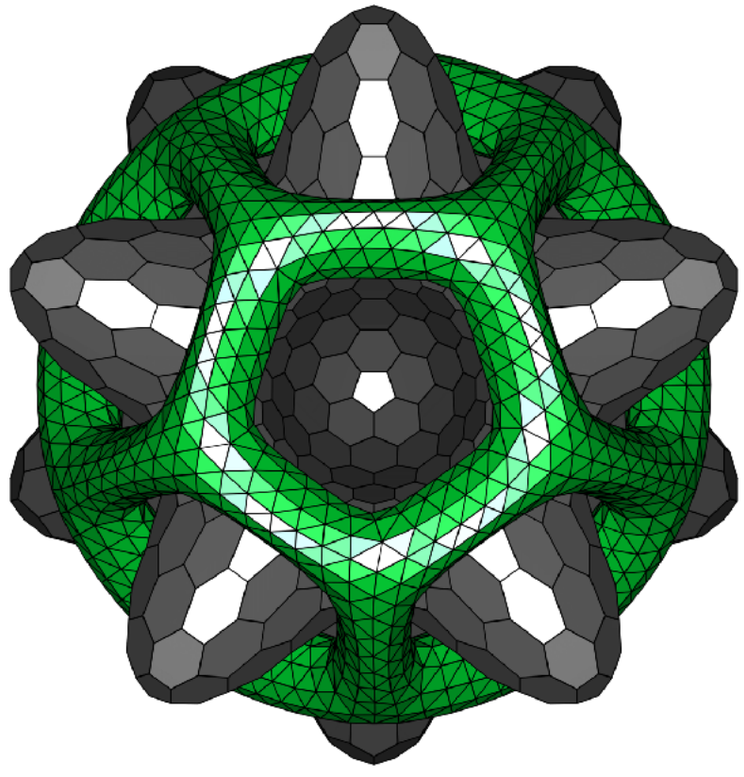


JOSS blog: http://blog.joss.theoj.org

Smith et al. (2018), Journal of Open Source Software (JOSS): design and first-year review. *PeerJ Comput. Sci*. 4:e147; DOI 10.7717/peerj-cs.147

Sister Journal: JOSE
Journal of Open Source Education

- Part of the Open Journals, sibling journal to JOSS (literally forked from it!). Also NumFOCUS affiliate.
- JOSE publishes two types of articles that describe:
  - ✴open educational software tools
  - ✴open-source learning modules
- Motivation: credit efforts to develop software for assisting teaching/learning and open-source educational content
- Submissions are peer-reviewed, with the intent of *improving the quality of the software or content submitted*



JOSE Editorial Board

Lorena Barba
George Washington Univ.

Katy Huff
UIUC

Juan Flopped
University of Cape Town

Jason Moore
UC Davis

Kyle Niemeyer
Oregon State

Charles Severance
Univ. Michigan

Robert Talbert
Grand Valley State U.

Tracy Teal
Univ. Michigan

Carol Willing
Cal Poly San Luis Obispo

Summary and Future Work

Summary and Future Work

- Course offered twice with ME 599 temporary class identifier; plan to offer again in Spring 2021—will need permanent “location”.

Summary and Future Work

- Course offered twice with ME 599 temporary class identifier; plan to offer again in Spring 2021—will need permanent “location”.

- Most students come in with little/no Unix command line experience, and only some with Python programming—plan to offer regular Software Carpentry workshops for first-year grad students in engineering.

Summary and Future Work

- Course offered twice with ME 599 temporary class identifier; plan to offer again in Spring 2021—will need permanent “location”.

- Most students come in with little/no Unix command line experience, and only some with Python programming—plan to offer regular Software Carpentry workshops for first-year grad students in engineering.

- Developing standalone modules for topics, to be shared openly

Summary and Future Work

- Course offered twice with ME 599 temporary class identifier; plan to offer again in Spring 2021—will need permanent “location”.

- Most students come in with little/no Unix command line experience, and only some with Python programming—plan to offer regular Software Carpentry workshops for first-year grad students in engineering.

- Developing standalone modules for topics, to be shared openly

- Submit to JOSS and/or JOSE!

Thank you!

Feedback and questions are welcome.

softwaredevengresearch.github.io/syllabus-s2019

niemeyer-research-group.github.io

Funding: Better Scientific Software Fellowship, part of the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.