A Project-Based Course on Software Development for (Engineering) Research

Kyle E. Niemeyer(✉)

Oregon State University, Corvallis, OR 97331, USA
kyle.niemeyer@oregonstate.edu

Abstract. This paper describes the motivation and design of a 10-week graduate course that teaches practices for developing research software; although offered by an engineering program, the content applies broadly to any field of scientific research where software may be developed. Topics taught in the course include local and remote version control, licensing and copyright, structuring Python modules, testing and test coverage, continuous integration, packaging and distribution, open science, software citation, and reproducibility basics, among others. Lectures are supplemented by in-class activities and discussions, and all course material is shared openly via GitHub. Coursework is heavily based on a single, term-long project where students individually develop a software package targeted at their own research topic; all contributions must be submitted as pull requests and reviewed/merged by other students. The course was initially offered in Spring 2018 with 17 students enrolled, and will be taught again in Spring 2019.

Keywords: Research software · Teaching software development · Software best practices

1 Motivation

Nearly all research relies on software—even experimental—but researchers typically do not receive training in best practices during graduate school in the same way as they do for experimental methods. In fact, in two recent surveys the vast majority of academics confirmed that they use software and that their research would be impractical without it: 90%/70% of UK academics surveyed in 2014 [12], and 95%/63% of US postdoctoral researchers surveyed in 2017 [19]. Computational science in particular depends on software and following good, evidence-based practices when working with software and data.

For example, in the Mechanical Engineering graduate curriculum at Oregon State University, the thermal-fluid sciences option (where I teach) requires a course on experimental measurement techniques, but no analogous course on proper techniques on software development or computational science. (We do require a course on numerical methods that focuses on solving differential equations, but this does not extend to software development.) Instead, our program—as in most similar programs around the world—assumes that such practices

J. M. F. Rodrigues et al. (Eds.): ICCS 2019, LNCS 11540, pp. 401–407, 2019.
https://doi.org/10.1007/978-3-030-22750-0_32

are trivial compared with the physical phenomena or mathematical methods and/or can be self-taught. However, in the same way that appropriate measurement techniques and statistical analysis of data are necessary for experimental (and computational) research, good practices ensure reliability and correctness of research results obtained from software developed for both computational (i.e., modeling-based) or experimental (i.e., analysis of results) research.

As the research community has recognized the importance of software and data skills, in recent years Software Carpentry [27] workshops have become a recognized avenue for graduate students and postdocs (and the occasional faculty member) to learn necessary skills for working with Python, the command line, and version-control systems. While these are essential skills for research, researchers who go further to develop software require additional training. This article describes a course aimed at filling this gap by teaching graduate-student researchers practical software development skills, and also exposing them to topics related to open science and reproducibility.

2 Course Design

The course heavily relies and builds on the *Effective Computation in Physics* textbook by Scopatz and Huff [23] (Chaps. 10–22), as well as recommendations by Wilson et al. [28] and Jiménez et al. [13]. All materials for the course are openly available online via GitHub and shared under a Creative Commons Attribution license; the online course syllabus provides links to each lecture [20]. The course combines lectures, hosted on GitHub and presented using `reveal.js` [8], with in-class activities and discussions, as summarized in Table 1. Out-of-class work, described in Table 2, centers around a software development project discussed in Sect. 2.2.

2.1 Course Description and Learning Objectives

The listed course description is

> This course will advance students' understanding of topics related to computational science and engineering, and advance their skills in applying techniques to solve research problems using high-level programming languages. The course will build on existing abilities in computer programming to cover topics related to computational modeling and scientific software development. Students will gain experience in applying available packages and libraries, as well as developing software to solve problems related to their own research interests. Students will also gain experience in working collaboratively and openly on scientific computing projects.

Table 1. Course schedule over ten weeks, with in-class activities.

Topic	In-class activity
Getting started, and version control	Configure Git
Remote version control, licensing, and copyright	Create, clone, and fork repos
Structuring modules, and testing	Create basic structure of module
Test coverage, continuous integration, documentation	Configure Travis CI
Introduction to Julia (guest lecture)	
Introduction to parallel programming	
Classes and objects (in Python)	
Packaging and distributing your software	Create PyPI, Anaconda packages
Optimizing numerical code in Python	
Working with files, command-line inputs in Python	
Open science, software citation, reproducibility	Connect GitHub and Zenodo
Posters, presentations, and technical writing	
Project presentations	

Table 2. Assignments, with week assigned.

Assignment	Week
Join Gitter chat room and create GitHub profile	1
Project proposal	1
Choose open-source license	3
Create tests and submit as PR for review	3
Finish configuring Travis CI	4
Write comments and docstrings	4
Complete PyPI and/or Anaconda packages	6
Write report and make presentation	7

By the end of the course, students should be able to

1. use high-level programming language to analyze and/or solve practical research problems;
2. apply principles of modern computational science and engineering, reproducibility, and open science to their research;
3. evaluate, visualize, write about, and publish computational research results; and

4. develop and share an open-source research software package that solves a problem in their research area.

These are the formal student learning objectives for the course.

2.2 Project

In lieu of standalone homework assignments, all assigned work contributes to a term-long project where students develop a new software package targeted at their own research area. The project initiates with a proposal that students submit via pull request to an open repository on GitHub, and which the instructor merges upon approval (following any changes requested). Then, students create a repository in the course organization for their software package, and fork this to their own accounts. After this, students submit all project contributions as pull requests to the upstream repository. Partners review these and either approve or request changes; only after the code-review partner approves the contribution can the project's owner merge the pull request.

2.3 Methods of Instruction

The course is delivered using a combination of lectures, discussion, and in-class interactive work. Lectures mostly exist as `reveal.js` [9] presentations, which are shared openly on a public-facing syllabus website [20]. Lectures also use practical demonstrations of Python code, shown using either IPython [21] or via Jupyter Notebooks [15]. Nearly all lectures also ask students to follow along on their own computer, either executing example code or advancing their project packages.

3 Results from First Offering

I offered the first iteration of this course in the Spring 2018 term, with the title "Software Development for Engineering Research;" while the course content is not limited to engineering research, I offered the course out of the Mechanical Engineering program with a targeted audience of graduate students in the College of Engineering. 17 students enrolled in the course, with all but one being graduate students; roughly half were in the second year or later of their graduate programs. Approximately 40% of the students came from mechanical engineering (including thermal-fluid sciences and design engineering), 35% were from nuclear engineering, and the remaining came from robotics and chemical engineering. Three quarters of the students had already taken a course on Python programming for engineering applications, while the others had some self-taught Python programming skills. Half expressed comfort working with the Unix command line, and the other half said they had used it but were not as comfortable with command-line operations. None admitted to being command-line ninjas, and none were completely unfamiliar.

The first offering of this 10-week course on software development for engineering research completed successfully in June 2018, with all 17 students releasing

the first version of their software developed during the course. At least four of the software packages have been developed further after the conclusion of the course, and at least one package is being prepared for submission to the *Journal of Open Source Software* (JOSS) [24].

The projects covered a wide variety of topics, with functions including simulation, experimental data analysis, and automation: designing detonation tubes [6], using machine learning to extract features from nuclear physics simulations [10], interfacing with an 8-channel digital pulse processor board [16], simulating and analyzing the combustion engine of a Global Formula Racing formula SAE vehicle [14], optimizing and analyzing wind-farm layouts [17], analyzing spin stabilization of solid rocket motors [18], a nodal quasi-diffusion solver for nuclear fission [22], agent-learning for autonomous path finding [25], generating input for a Monte-Carlo radiation transport code [26], calculating solar-energy terms based on location [11], analyzing radioxenon spectra [7], calculating deep-learning layers for multi-agent reinforcement learners [5], analyzing solvent extraction kinetics [4], simulating rapid compression machine experiments [3], calibrating blackbody infrared cameras [2], and simulating transient heat transfer in a microchannel with passive temperature dependent flow control [1].

Although the sample size is small, students rated the course well in their end-of-term evaluations: they rated the course as a whole 5.3/5.5 (mean/median) out of 6.0 and the instructor contribution 5.5/5.8 (mean/median) out of 6.0. Multiple comments discussed the course favorably, and that it should be taken by all students doing research involving software/programming. Suggestions included clarifying expectations for students and proving more feedback; also, one student felt the course was too advanced for their experience level.

4 Conclusions

This article describes a 10-week course given in Spring 2018 teaching skills for developing research software; all lesson and assignment content is available openly [20]. This course will be offered again in Spring term 2019 (10 weeks, April–June). Planned changes include incorporating more in-class activities in more of the topics, and adding new topics such as peer code review and high-performance computing.

In addition, I am developing alternate versions of the course aimed at different lengths of time, such as an afternoon tutorial session or day-long workshop. These lessons and modules will be shared openly for the community to use, adapt, and extend. Furthermore, while the course at Oregon State University is currently offered out of the Mechanical Engineering program, it may fit better offered as an Engineering course or more broadly in the Graduate Education program.

Acknowledgements. This research was supported by the 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.

References

1. Armatis, P.D.: `PAFloCS` v0.1.0. Zenodo (2018). https://doi.org/10.5281/zenodo.1291277
2. Bean, D.: `IRCal` v0.1.0. Zenodo (2018). https://doi.org/10.5281/zenodo.1291190
3. Behnoudfar, D.: `SimRCM` v0.1.0. Zenodo (2018). https://doi.org/10.5281/zenodo.1291302
4. Bettinardi, D.J.: `SepKinetics` pre-alpha v0.110. Zenodo (2018). https://doi.org/10.5281/zenodo.1291202
5. Brian, M.: `deep_learning_layer_calculator` v1.0. Zenodo (2018). https://doi.org/10.5281/zenodo.1291273
6. Carter, M.: `BeaverDet` v0.1.0. Zenodo (2018). https://doi.org/10.5281/zenodo.1288009
7. Czyz, S.A.: `radioxenon_ml` v0.5.0. Zenodo (2018). https://doi.org/10.5281/zenodo.1291208
8. El Hattab, H.: `reveal.js` 3.7.0, December 2018. https://github.com/hakimel/reveal.js
9. El Hattab, H.: `reveal.js` (2019). https://github.com/hakimel/reveal.js
10. Grechanuk, P.A.: `ML_FeatureAnalysis` v0.0.1. Zenodo (2018). https://doi.org/10.5281/zenodo.1286598
11. Guymer, N.: `SolarCalc` v0.1.0. Zenodo (2018). https://doi.org/10.5281/zenodo.1290030
12. Hettrick, S., et al.: UK research software survey 2014 (dataset). University of Edinburgh on behalf of Software Sustainability Institute (2015). https://doi.org/10.7488/ds/253
13. Jiménez, R.C., et al.: Four simple recommendations to encourage best practices in research software. F1000Research **6**, 876 (2017). https://doi.org/10.12688/f1000research.11407.1
14. Kittelman, J.S.: `PyPow` v0.0.1. Zenodo (2018). https://doi.org/10.5281/zenodo.1287325
15. Kluyver, T., et al.: Jupyter notebooks - a publishing format for reproducible computational workflows. In: Loizides, F., Schmidt, B. (eds.) Positioning and Power in Academic Publishing: Players, Agents and Agendas, pp. 87–90. IOS Press (2016). https://doi.org/10.3233/978-1-61499-649-1-87
16. Mannino, M.: `OkPy_Rad`. GitHub (2018). https://github.com/SoftwareDevEngResearch/OKPY_Rad
17. Miller, A.: `BigFan` v0.1.2. Zenodo (2018)
18. Morse, M.D.: `SRMspinanalysis` v0.1.0. Zenodo (2018). https://doi.org/10.5281/zenodo.1288020
19. Nangia, U., Katz, D.S.: Track 1 paper: surveying the U.S. National Postdoctoral Association regarding software use and training in research. In: Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE 5.1) (2017). https://doi.org/10.6084/m9.figshare.5328442.v3
20. Niemeyer, K.E.: Syllabus for Software Development for Engineering Research course (2018). https://softwaredevengresearch.github.io/syllabus/
21. Pérez, F., Granger, B.E.: IPython: a system for interactive scientific computing. Comput. Sci. Eng. **9**(3), 21–29 (2007). https://doi.org/10.1109/MCSE.2007.53. https://ipython.org
22. Reynolds, A.J.: `NoQuD`. Zenodo (2018). https://doi.org/10.5281/zenodo.1290779

23. Scopatz, A., Huff, K.D.: Effective Computation in Physics: Field Guide to Research with Python. O'Reilly Media, Inc. (2015). http://physics.codes
24. Smith, A.M., et al.: Journal of Open Source Software (JOSS): design and first-year review. PeerJ Comput. Sci. **4**, e147 (2018). https://doi.org/10.7717/peerj-cs.147
25. Sripada, V.: `agent_learning`. Zenodo (2018). https://doi.org/10.5281/zenodo.1291474
26. Stewart, R.: `FRIDGe` v0.1.0-alpha. Zenodo (2018). https://doi.org/10.5281/zenodo.1288681
27. Wilson, G.: Software carpentry: lessons learned. F1000Research **3**, 62 (2016). https://doi.org/10.12688/f1000research.3-62.v2
28. Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., Teal, T.K.: Good enough practices in scientific computing. PLOS Comput. Biol. **13**(6), e1005510 (2017). https://doi.org/10.1371/journal.pcbi.1005510