
JOSE Documentation

Open Journals

Sep 25, 2019

1	About this site	3
2	Submitting a paper to JOSE	5
3	Sponsors and affiliates	7
3.1	Submitting a paper to JOSE	7
3.1.1	The two submission types	7
3.1.2	Submission requirements	8
3.1.3	What about novelty?	8
3.1.4	How to prepare a software submission?	8
3.1.5	How to prepare a learning-module submission?	8
3.1.6	What should my paper contain?	8
3.1.7	Example paper and bibliography	9
3.1.8	Submitting your paper	12
3.1.9	No submission fees	12
3.1.10	Preprint Policy	12
3.1.11	Authorship	12
3.1.12	Submissions using proprietary languages/dev environments	12
3.1.13	The review process	13
3.2	Reviewing for JOSE	13
3.2.1	JOSE review principles	13
3.2.2	JOSE Conflict of Interest Policy	13
3.2.3	JOSE aim and scope	14
3.3	Review criteria	14
3.3.1	The JOSE paper	14
3.3.2	Submission requirements	15
3.3.2.1	License	15
3.3.2.2	Statement of need	15
3.3.2.3	Community guidelines	15
3.3.2.4	Specific requirements for software submissions	15
3.3.2.5	Specific requirements for learning modules	15
3.3.3	Other considerations	16
3.3.3.1	Authorship	16
3.3.3.2	What happens if the submission I'm reviewing doesn't meet the JOSE criteria?	16
3.4	Review checklist	16
3.4.1	Conflict of interest	16
3.4.2	Code of Conduct	16

3.4.3	General checks	17
3.4.4	Documentation	17
3.4.5	Pedagogy / Instructional design (Work-in-progress: reviewers, please comment!)	17
3.4.6	JOSE paper	17
3.5	Editorial Guide	18
3.5.1	Pre-review	18
3.5.1.1	How papers are assigned to editors	18
3.5.1.2	Finding reviewers	18
3.5.1.3	Starting the review	19
3.5.2	Review	19
3.5.3	After acceptance	19
3.5.4	Processing of rOpenSci-reviewed and accepted submissions	20
3.5.5	Sample letter to invite reviewers	20
3.5.6	Overview of editorial process	21
3.5.7	Visualization of editorial flow	23
3.5.8	Expectations on JOSS editors	23
3.5.8.1	Responding to editorial assignments	23
3.5.8.2	Continued attention to assigned submissions	24
3.5.9	Out of office	24
3.6	Interacting with Whedon	24
3.6.1	Author commands	25
3.6.1.1	Compiling papers	25
3.6.1.1.1	Compiling papers from a non-default branch	26
3.6.1.2	Finding reviewers	26
3.6.2	Editorial commands	26
3.6.2.1	Assigning an editor	26
3.6.2.2	Adding and removing reviewers	26
3.6.2.3	Starting the review	27
3.6.2.4	Reminding reviewers and authors	27
3.6.2.5	Setting the software archive	27
3.6.2.6	Changing the software version	27
3.6.3	Accepting a paper (dry run)	28
3.6.3.1	Check references	28
3.6.4	Accepting a paper (for real)	28
3.7	Installing the JOSE application	28

The [Journal of Open Source Education](#) (JOSE) is an **educator-friendly** journal for publishing computational learning modules and educational software.

JOSE, pronounced [hoe-zay], is an academic journal (ISSN 2577-3569) and sibling to the [Journal of Open Source Software](#), which publishes open research software. JOSE relies on the journal management infrastructure and tools developed for JOSS. Upon acceptance into JOSE, we mint a CrossRef DOI for your paper and we list it on the [JOSE website](#).

JOSE publishes two types of articles that describe:

- open educational software tools
- open-source learning modules

CHAPTER 1

About this site

This site contains documentation for authors interested in submitting to JOSE, reviewers who have generously volunteered their time to review submissions, and editors who manage the JOSE editorial process.

If you're interested in learning more about JOSE, you might want to read:

- The [about page](#) on the main JOSE site

CHAPTER 2

Submitting a paper to JOSE

If you'd like to submit a paper to JOSE, please read the author submission guidelines in the *Submitting a paper to JOSE* section.



JOSE is a proud affiliate of the [Open Source Initiative](#). As such, we are committed to public support for open source software and the role OSI plays therein. In addition, Open Journals (the parent entity behind JOSE) is a [NumFOCUS](#)-sponsored project.

3.1 Submitting a paper to JOSE

Preparing your JOSE submission should be a simple task, once you have a complete software or learning module you wish to publish.

3.1.1 The two submission types

JOSE accepts two types of submissions: (1) computational learning modules, created as open educational resources (OER), and (2) open-source software, created as educational technology or infrastructure.

3.1.2 Submission requirements

JOSE submissions must be fully open, under the [Open Definition](#). This means that any text content or graphical objects should be under a Creative Commons license (ideally [CC-BY](#)) and code components should be under an [OSI-approved license](#).

Computational learning modules should be complete and immediately usable for self-learning or adoption by other instructors. They should make a clear contribution to teaching and learning of any subject, supported by computing. JOSE is not focused in OER for “learning to code” as much as “coding to learn.”

Software submissions should make a clear contribution to the available open-source software that supports teaching and learning, or makes an educational process better (e.g., faster, easier, simpler). Examples include software to auto-grade student work, learning management systems, and student collaboration frameworks. Software should be feature-complete (no half-baked solutions).

3.1.3 What about novelty?

Authors make the case for their submission’s contribution in the paper, under the heading “Statement of Need.” The criterion is less one of novelty, than need: submissions targeting subjects or applications already addressed by other resources are eligible, if the authors make a case for why they might be adopted by learners or other instructors. For example, authors could say that they approach a topic differently, that they update material to modern contexts, that the need is born of a new educational program, or a conference tutorial or other informal-learning scenario.

3.1.4 How to prepare a software submission?

Before starting your submission, you should:

- Have the software available in an open repository (GitHub, Bitbucket, etc.) under an [OSI-approved license](#).
- Write a short Markdown file titled `paper.md` with title, author names and affiliations, containing a description of the software, statement of need, and key references.
- References should be included in a BibTeX file called `paper.bib`

Once you have those items in place, [submit](#) via the JOSE web app.

3.1.5 How to prepare a learning-module submission?

Before starting your submission, you should:

- Have the content in an open repository, under a Creative Commons license (ideally [CC-BY](#)), and any code components under an [OSI-approved license](#).
- Write a short Markdown file titled `paper.md` with title, author names and affiliations, containing a description of the module, a summary of its contents, a statement of need, and key references.
- References should be included in a BibTeX file called `paper.bib`

Once you have those items in place, [submit](#) via the JOSE web app.

3.1.6 What should my paper contain?

JOSE papers should:

- List all authors and affiliations.

- Describe the submission, and explain its eligibility for JOSE.
- Include a “Statement of Need” section, explaining how the submitted artifacts contribute to computationally enabled teaching and learning, and describing how they might be adopted by others.
- For software submissions, describe the functionality of the software, usage and recent experience of use in teaching and learning situations.
- For learning modules, describe the learning objectives, content, instructional design, and experience of use in teaching and learning situations.
- Tell us the “story” of the project: how did it come to be?
- Cite key references, including a link to the open archive of the software or the learning module.

JOSE welcomes submissions with diverse educational contexts. You should write your paper for a non-specialist reader. Your submission should probably be around 1000 words (or around two pages).

The goal is that someone reading the JOSE paper has enough information to decide if they’d be interested in adopting the learning module or software. Readers will want to know how the content/software has been used, and how they would adopt it. They may also want to be persuaded that the authors have put careful work on creating the material, and have experience teaching with it.

JOSE papers contain a limited set of metadata, plus Summary & Reference sections. We explicitly do not publish long-form articles, because the scholarship represented by a JOSE article is contained in the software or learning modules themselves. Expected length is around 1000 words max.

3.1.7 Example paper and bibliography

This example `paper.md` is adapted from the JOSS paper *Gala: A Python package for galactic dynamics* by Adrian M. Price-Whelan <http://doi.org/10.21105/joss.00388>:

```
---
title: 'Gala: A Python package for galactic dynamics'
tags:
  - Python
  - astronomy
  - dynamics
  - galactic dynamics
  - milky way
authors:
  - name: Adrian M. Price-Whelan
    orcid: 0000-0003-0872-7098
    affiliation: "1, 2" # (Multiple affiliations must be quoted)
  - name: Author Without ORCID
    affiliation: 2
affiliations:
  - name: Lyman Spitzer, Jr. Fellow, Princeton University
    index: 1
  - name: Institution 2
    index: 2
date: 13 August 2017
bibliography: paper.bib

# Optional fields if submitting to a AAS journal too, see this blog post:
# https://blog.joss.theoj.org/2018/12/a-new-collaboration-with-aas-publishing
aas-doi: 10.3847/xxxxx <- update this with the DOI from AAS once you know it.
aas-journal: Astrophysical Journal <- The name of the AAS journal.
---
```

(continues on next page)

```

# Summary

The forces on stars, galaxies, and dark matter under external gravitational
fields lead to the dynamical evolution of structures in the universe. The orbits
of these bodies are therefore key to understanding the formation, history, and
future state of galaxies. The field of "galactic dynamics," which aims to model
the gravitating components of galaxies to study their structure and evolution,
is now well-established, commonly taught, and frequently used in astronomy.
Aside from toy problems and demonstrations, the majority of problems require
efficient numerical tools, many of which require the same base code (e.g., for
performing numerical orbit integration).

``Gala`` is an Astropy-affiliated Python package for galactic dynamics. Python
enables wrapping low-level languages (e.g., C) for speed without losing
flexibility or ease-of-use in the user-interface. The API for ``Gala`` was
designed to provide a class-based and user-friendly interface to fast (C or
Cython-optimized) implementations of common operations such as gravitational
potential and force evaluation, orbit integration, dynamical transformations,
and chaos indicators for nonlinear dynamics. ``Gala`` also relies heavily on and
interfaces well with the implementations of physical units and astronomical
coordinate systems in the ``Astropy`` package [@astropy] (``astropy.units`` and
``astropy.coordinates``).

``Gala`` was designed to be used by both astronomical researchers and by
students in courses on gravitational dynamics or astronomy. It has already been
used in a number of scientific publications [@Pearson:2017] and has also been
used in graduate courses on Galactic dynamics to, e.g., provide interactive
visualizations of textbook material [@Binney:2008]. The combination of speed,
design, and support for Astropy functionality in ``Gala`` will enable exciting
scientific explorations of forthcoming data releases from the *Gaia* mission
[@gaia] by students and experts alike.

# Mathematics

Single dollars ($) are required for inline mathematics e.g.  $f(x) = e^{\pi/x}$ 

Double dollars make self-standing equations:


$$\Theta(x) = \begin{array}{l} 0 \text{ if } x < 0 \\ 1 \text{ else} \end{array}$$


# Citations

Citations to entries in paper.bib should be in
[rMarkdown] (http://rmarkdown.rstudio.com/authoring\_bibliographies\_and\_citations.html)
format.

For a quick reference, the following citation commands can be used:
- ``@author:2001`` -> "Author et al. (2001)"
- ``[@author:2001]`` -> "(Author et al., 2001)"
- ``[@author1:2001; @author2:2001]`` -> "(Author1 et al., 2001; Author2 et al., 2002)"

# Figures

```

(continues on next page)

(continued from previous page)

```
Figures can be included like this: ![Example figure.](figure.png)
```

```
# Acknowledgements
```

```
We acknowledge contributions from Brigitta Sipocz, Syrtis Major, and Semyeong Oh, and support from Kathryn Johnston during the genesis of this project.
```

```
# References
```

Example paper .bib file:

```
@article{Pearson:2017,
  Adsnote = {Provided by the SAO/NASA Astrophysics Data System},
  Adsurl = {http://adsabs.harvard.edu/abs/2017arXiv170304627P},
  Archiveprefix = {arXiv},
  Author = {{Pearson}, S. and {Price-Whelan}, A.~M. and {Johnston}, K.~V.},
  Eprint = {1703.04627},
  Journal = {ArXiv e-prints},
  Keywords = {Astrophysics - Astrophysics of Galaxies},
  Month = mar,
  Title = {{Gaps in Globular Cluster Streams: Pal 5 and the Galactic Bar}},
  Year = 2017
}

@book{Binney:2008,
  Adsnote = {Provided by the SAO/NASA Astrophysics Data System},
  Adsurl = {http://adsabs.harvard.edu/abs/2008gady.book....B},
  Author = {{Binney}, J. and {Tremaine}, S.},
  Booktitle = {Galactic Dynamics: Second Edition, by James Binney and Scott
↪Tremaine.~ISBN 978-0-691-13026-2 (HB).~Published by Princeton University Press,
↪Princeton, NJ USA, 2008.},
  Publisher = {Princeton University Press},
  Title = {{Galactic Dynamics: Second Edition}},
  Year = 2008
}

@article{gaia,
  author = {{Gaia Collaboration}},
  title = "{The Gaia mission}",
  journal = {\aap},
  archivePrefix = "arXiv",
  eprint = {1609.04153},
  primaryClass = "astro-ph.IM",
  keywords = {space vehicles: instruments, Galaxy: structure, astrometry,
↪parallaxes, proper motions, telescopes},
  year = 2016,
  month = nov,
  volume = 595,
  doi = {10.1051/0004-6361/201629272},
  adsurl = {http://adsabs.harvard.edu/abs/2016A%26A...595A...1G},
}

@article{astropy,
  author = {{Astropy Collaboration}},
  title = "{Astropy: A community Python package for astronomy}",
  journal = {\aap},
```

(continues on next page)

```
archivePrefix = "arXiv",
eprint = {1307.6212},
primaryClass = "astro-ph.IM",
keywords = {methods: data analysis, methods: miscellaneous, virtual observatory_
→tools},
year = 2013,
month = oct,
volume = 558,
doi = {10.1051/0004-6361/201322068},
adsurl = {http://adsabs.harvard.edu/abs/2013A%26A...558A..33A}
}
```

Note that the paper ends with a References heading, and the references are built automatically from the content in the `.bib` file. You should enter in-text citations in the paper body following correct [Markdown citation syntax](#).

3.1.8 Submitting your paper

Submission is as simple as:

- Filling in the [short submission form](#)
- Waiting for the managing editor to start a pre-review issue over in the JOSS reviews repository: <https://github.com/openjournals/jose-reviews>

3.1.9 No submission fees

There are no fees for submitting or publishing in JOSE. You can read more about our [cost and sustainability model](#).

3.1.10 Preprint Policy

Authors are welcome to submit their papers to a preprint server ([arXiv](#), [bioRxiv](#), [SocArXiv](#), [PsyArXiv](#) etc.) at any point during the submission and review process.

Submission to a preprint server is *not* considered a previous publication.

3.1.11 Authorship

Purely financial (such as being named on an award) and organizational (such as general supervision of a research group) contributions are not considered sufficient for co-authorship of JOSE submissions, but active project direction and other forms of non-code contributions are. The authors themselves assume responsibility for deciding who should be credited with co-authorship, and co-authors must always agree to be listed. In addition, co-authors agree to be accountable for all aspects of the work, and to notify JOSE if any retraction or correction of mistakes are needed after publication.

3.1.12 Submissions using proprietary languages/dev environments

We strongly prefer software that doesn't rely upon proprietary (paid for) development environments/programming languages. However, provided *your submission meets our requirements* (including having a valid open source license) then we will consider your submission for review. Should your submission be accepted for review, we may ask you, the submitting author, to help us find reviewers who already have the required development environment installed.

3.1.13 The review process

We encourage you to familiarize yourself with our [review criteria](#) as this will help you understand what our reviewers will be looking for. Broadly speaking though, provided you have followed our pre-submission steps and meet our submission requirements then you should expect a rapid review (typically less than two weeks).

After submission:

- The Editor-in-Chief will carry out an initial check of your submission, and proceed to assign a handling editor.
- The handling editor will assign two or more JOSE reviewers, and the review will be carried out in the [JOSE reviews repository](#).
- Authors will respond to reviewer-raised issues (if any are raised) on the submission repository's issue tracker. Reviewer and editor contributions, like any other contributions, should be acknowledged in the repository.
- Upon successful completion of the review, deposit a copy of your (updated) repository with a data-archiving service such as [Zenodo](#) or [figshare](#), get a DOI for the archive, and update the review issue thread with the DOI.
- After we assign a DOI for your accepted JOSE paper, its metadata is deposited with CrossRef and listed on the JOSE website.
- The review issue will be closed, and an automatic tweet from [@JOSE_TheOJ](#) will announce it!

If you want to learn more details about the review process, take a look at the [reviewer guidelines](#).

3.2 Reviewing for JOSE

Firstly, thank you so much for agreeing to review for the Journal of Open Source Education (JOSE)—we're delighted to have your help! Follow this guide to help you complete your review, and feel free to ask editors for more help using any medium (email, Twitter, comments on review issues, etc.)

3.2.1 JOSE review principles

JOSE accepts two types of submissions, and your role as reviewer will be different depending on the type. For software submissions, you will download, build, test and assess the software quality. For learning modules, you will download and read, test code components, and assess content and pedagogy.

For both types of submissions, your goal as reviewer is to help authors improve the quality of the submission to the standard for acceptance into JOSE. Categorical rejections are expected only in cases of out-of-scope or otherwise ineligible submissions. If authors decide they cannot meet your requests for improvements, however, they can withdraw their submission.

JOSE provides a checklist for review. You will check off each item, as you proceed with the review, adding comments and suggestions in the Review issue thread. You may also post specific issues on the main repository of the submitted software or module. Be sure to post links between the JOSE Review issue and the submission repository's issue tracker, as needed.

3.2.2 JOSE Conflict of Interest Policy

The definition of a conflict of Interest in peer review is a circumstance that makes you “unable to make an impartial scientific judgment or evaluation.” ([PNAS Conflict of Interest Policy](#)). JOSE is concerned with avoiding any actual conflicts of interest, and being sufficiently transparent that we avoid the appearance of conflicts of interest as well.

As a reviewer, COIs are your present or previous association with any authors of a submission: recent (past four years) collaborators in funded research or work that is published; and lifetime for the family members, business partners,

and thesis student/advisor or mentor. In addition, your recent (past year) association with the same organization of a submitter is a COI, for example, being employed at the same institution.

If you have a conflict of interest with a submission, you should disclose the specific reason to the submissions' editor. This may lead to you not being able to review the submission, but some conflicts may be recorded and then waived, and if you think you are able to make an impartial assessment of the work, you should request that the conflict be waived. For example, if you and a submitter were two of 2000 authors of a high energy physics paper but did not actually collaborate. Or if you and a submitter worked together 6 years ago, but due to delays in the publishing industry, a paper from that collaboration with both of you as authors was published 2 year ago. Or if you and a submitter are both employed by the same very large organization but in different units without any knowledge of each other.

Declaring actual, perceived, and potential conflicts of interest is required under professional ethics. If in doubt: ask the editors.

3.2.3 JOSE aim and scope

We're witnessing a swelling of communities that develop and share tools and practices in computationally enabled teaching and learning. The founding editors of JOSE have been participating in these communities, and recognized a need for a publication to house these scholarly products in a citable form.

Several journals already exist that publish academic papers reporting on *education research*: the systematic collection and analysis of data on teaching methods and student learning, and related qualitative studies. JOSE has a different scope. We're focused on the rich and growing work applying computational methods to enhance teaching and learning. For example, a whole new genre of open educational resources (OER) has sprouted out of creating and sharing Jupyter notebooks, in a variety of subjects benefitting from a computational treatment. At the same time, members of open-source communities are creating software tools to automate processes (like grading), facilitate access to computing for learning (via cloud systems), or otherwise enhance teaching and learning with computing. These are valuable contributions, independently of their possible use in education research, and their creators merit publication and citation credit for them.

JOSE also aims to disseminate the best practices in creating these scholarly objects, and increase quality via peer review. We want to be a formal, yet open and collegial vehicle for sharing accumulated knowledge in using computing to teach and learn.

3.3 Review criteria

3.3.1 The JOSE paper

As outlined in the [submission guidelines provided to authors](#), the JOSE paper (the compiled PDF associated with this submission) should only include:

- Title, and author list with affiliations
- Description of the software or learning module
- Statement of need
- Key references, including the submission archive

Detailed documentation should be present in the repository of the submitted software or module, is reviewed there, and does not need to be included in the paper.

Important: Note that software paper's *should not* include software documentation such as API (Application Programming Interface) functionality, as this should be outlined in the software documentation.

3.3.2 Submission requirements

Important: Note, if you've not yet been involved in a JOSE review, you can see an example JOSE review checklist [here](#).

3.3.2.1 License

The submission should be open, under the [Open Definition](#). Any text content or graphical objects should be under a Creative Commons license (ideally [CC-BY](#)) and code components should be under an [OSI-approved license](#). License information must be clearly visible in the submission's online repository, which must include a plain-text LICENSE file.

Acceptable: A plain-text LICENSE file with the contents of an OSI-approved license **Not acceptable:** A phrase such as 'MIT license' in a README file

3.3.2.2 Statement of need

A key component of the JOSE paper is a statement by the authors, explaining the contribution made by the submitted artifacts to computationally enabled teaching and learning, and describing how they might be used by others. The criterion is less one of novelty, than *need*: submissions targeting subjects or applications already addressed by other resources are *eligible*, if the authors make a case for why they might be adopted by learners or other instructors.

3.3.2.3 Community guidelines

The online repository of the software or learning module needs to contain clear guidelines for potential contributors who may want to:

- Contribute to the software/module
- Report issues or problems with the software/module
- Seek support

3.3.2.4 Specific requirements for software submissions

Documentation: The software repository should contain enough documentation to understand the functionality of the software, to guide through the build process (including a list of dependencies), and to complete examples of use.

Tests: Software quality depends on testing. Ideally, the software should include an automated test suite, but it's also acceptable to include documented manual steps to test the functionality of the software.

Examples: Potential users of new software rely on well-documented examples to get started. Reviewers will look for examples of use that illustrate beginner and advanced functionality

Good: A package management file such as a Gemfile or package.json or equivalent **OK:** A list of dependencies to install **Bad (not acceptable):** Reliance on other software not listed by the authors

3.3.2.5 Specific requirements for learning modules

Substance: A learning module should cover a substantial portion of material to achieve plainly clear learning objectives. The ideal module consists of a few lessons, building up a well-rounded topic, as a full tutorial or *part* of a term or semester course. This direction follows trends and recommendations to 'modularize' courses, both online and

on-campus—one example is the 2014 report on the [Future of MIT Education](#). The module should contain well-written and complete presentation of the material, weaved with the computational portions and sample output.

Pedagogical soundness: Instructional design of the module should be intentional and apparent. For example, the weaving of text, images, code and output might apply the [worked-example effect](#), deliberately. The authors should briefly explain their design in the JOSE paper.

3.3.3 Other considerations

3.3.3.1 Authorship

As part of the review process, you are asked to check whether the submitting author has made a “substantial contribution” to the submitted software (as determined by the commit history) and to check that “the full list of paper authors seems appropriate and complete?”

As discussed in the [submission guidelines for authors](#), authorship is a complex topic with different practices in different communities. Ultimately, the authors themselves are responsible for deciding which contributions are sufficient for co-authorship, although JOSE policy is that purely financial contributions are not considered sufficient. Your job as a reviewer is to check that the list of authors appears reasonable, and if it’s not obviously complete/correct, to raise this as a question during the review.

3.3.3.2 What happens if the submission I’m reviewing doesn’t meet the JOSE criteria?

We ask that reviewers grade submissions in one of three categories: 1) Accept 2) Minor Revisions 3) Major Revisions. Unlike some journals, we do not reject outright submissions requiring major revisions. We like to give the author as long as they need to make these modifications/improvements.

3.4 Review checklist

JOSE reviews are checklist-driven. That is, there is a checklist for each JOSE reviewer to work through when completing their review. A JOSE review is generally considered incomplete until the reviewer has checked off all of their checkboxes.

Below is an example of the review checklist.

Important: Note this section of our documentation only describes the JOSE review checklist. Authors and reviewers should consult the [review criteria](#) to better understand how these checklist items should be interpreted.

3.4.1 Conflict of interest

- As the reviewer I confirm that I have read the [JOSE conflict of interest policy](#) and that there are no conflicts of interest for me to review this work.

3.4.2 Code of Conduct

- I confirm that I read and will adhere to the [JOSE code of conduct](#).

3.4.3 General checks

- **Repository:** Is the source code for this learning module available at the repository url?
- **License:** Does the repository contain a plain-text LICENSE file with the contents of a standard license? (OSI-approved for code, Creative Commons for content)
- **Version:** Does the release version given match the GitHub release (v1.0)?
- **Authorship:** Has the submitting author made visible contributions to the module? Does the full list of authors seem appropriate and complete?

3.4.4 Documentation

- **A statement of need:** Do the authors clearly state the need for this module and who the target audience is?
- **Installation instructions:** Is there a clearly stated list of dependencies?
- **Usage:** Does the documentation explain how someone would adopt the module, and include examples of how to use it?
- **Community guidelines:** Are there clear guidelines for third parties wishing to 1) Contribute to the module 2) Report issues or problems with the module 3) Seek support?

3.4.5 Pedagogy / Instructional design (Work-in-progress: reviewers, please comment!)

- **Learning objectives:** Does the module make the learning objectives plainly clear? (We don't require explicitly written learning objectives; only that they be evident from content and design.)
- **Content scope and length:** Is the content substantial for learning a given topic? Is the length of the module appropriate?
- **Pedagogy:** Does the module seem easy to follow? Does it observe guidance on cognitive load? (working memory limits of 7 +/- 2 chunks of information)
- **Content quality:** Is the writing of good quality, concise, engaging? Are the code components well crafted? Does the module seem complete?
- **Instructional design:** Is the instructional design deliberate and apparent? For example, exploit worked-example effects; effective multi-media use; low extraneous cognitive load.

3.4.6 JOSE paper

- **Authors:** Does the `paper.md` file include a list of authors with their affiliations?
- **A statement of need:** Does the paper clearly state the need for this module and who the target audience is?
- **Description:** Does the paper describe the learning materials and sequence?
- Does it describe how it has been used in the classroom or other settings, and how someone might adopt it?
- Could someone else teach with this module, given the right expertise?
- Does the paper tell the “story” of how the authors came to develop it, or what their expertise is?
- **References:** Do all archival references that should have a DOI list one (e.g., papers, datasets, software)?

3.5 Editorial Guide

The Journal of Open Source Software (JOSS) conducts all peer review and editorial processes in the open, on the GitHub issue tracker.

JOSS editors manage the review workflow with the help of our bot, @whedon. The bot is summoned with commands typed directly on the GitHub review issues. For a list of commands, type: @whedon commands.

Note: To learn more about @whedon’s functionalities, take a look at our [dedicated guide](#).

3.5.1 Pre-review

Once a submission comes in, it will be in the queue for a quick check by the Editor-in-chief (EiC). From there, it moves to a PRE-REVIEW issue, where the EiC will assign a handling editor, and the author can suggest reviewers. Initial direction to the authors for improving the paper can already happen here, especially if the paper lacks some requested sections.

Important: If the paper is out-of-scope for JOSS, editors assess this and notify the author in the PRE-REVIEW issue.

The EiC assigns an editor (or a volunteering editor self-assigns) with the command @whedon assign @username as editor in a comment.

Note: If a paper is submitted without a recommended editor, it will show up in the weekly digest email under the category ‘Papers currently without an editor.’ Please review this weekly email and volunteer to edit papers that look to be in your domain. If you choose to be an editor in the issue thread type the command @whedon assign @yourhandle as editor

3.5.1.1 How papers are assigned to editors

By default, unless an editor volunteers, the Associated Editor-in-chief (AEiC) on duty will attempt to assign an incoming paper to the most suitable handling editor. While AEiCs will make every effort to match a submission with the most appropriate editor, there are a number of situations where an AEiC may assign a paper to an editor that doesn’t fit entirely within the editor’s research domains:

- If there’s no obvious fit to *any* of the JOSS editors
- If the most suitable editor is already handling a large number of papers
- If the chosen editor has a lighter editorial load than other editors

In most cases, an AEiC will ask one or more editors to edit a submission (e.g. @editor1, @editor 2 – would one of you be willing to edit this submission for JOSS). If the editor doesn’t respond within ~3 working days, the AEiC may assign the paper to the editor regardless.

3.5.1.2 Finding reviewers

At this point, the handling editor’s job is to identify reviewers who have sufficient expertise in the field of software and in the field of the submission. JOSS papers have to have a minimum of two reviewers per submission, except for papers that have previously been peer-reviewed via rOpenSci. In some cases, the editor also might want to formally

add themselves as one of the reviewers. If the editor feels particularly unsure of the submission, a third (or fourth) reviewer can be recruited.

To recruit reviewers, the handling editor can mention them in the PRE-REVIEW issue with their GitHub handle, ping them on Twitter, or email them. After expressing initial interest, candidate reviewers may need a longer explanation via email. See sample reviewer invitation email, below.

Once a reviewer accepts, the handling editor runs the command `@whedon assign @username as reviewer` in the PRE-REVIEW issue. Add more reviewers with the command `@whedon add @username as reviewer`.

Note: The `assign` command clobbers all reviewer assignments. If you want to add an additional reviewer use the `add` command.

3.5.1.3 Starting the review

Next, run the command `@whedon start review`. If you haven't assigned an editor and reviewer, this command will fail and `@whedon` will tell you this. This will open the REVIEW issue, with prepared review checklists for each reviewer, and instructions. The editor should close the PRE-REVIEW issue, at this point, and move the conversation to the separate REVIEW issue.

3.5.2 Review

The REVIEW issue contains some instructions, and reviewer checklists. The reviewer(s) should check off items of the checklist one-by-one, until done. In the meantime, reviewers can engage the authors freely in a conversation aimed at improving the paper.

If a reviewer recants their commitment or is unresponsive, editors can remove them with the command `@whedon remove @username as reviewer`. You can also add new reviewers in the REVIEW issue, but in this case, you need to manually add a review checklist for them by editing the issue body.

Comments in the REVIEW issue should be kept brief, as much as possible, with more lengthy suggestions or requests posted as separate issues, directly in the submission repository. A link-back to those issues in the REVIEW is helpful.

When the reviewers are satisfied with the improvements, we ask that they confirm their recommendation to accept the submission.

3.5.3 After acceptance

When a submission is accepted, we ask that the authors create an archive (on [Zenodo](#), [figshare](#), or other) and post the archive DOI in the REVIEW issue. The editor should run the command `@whedon set <archive doi> as archive`, and ping the EiC for final processing.

Steps:

- Get a new proof with the `@whedon generate pdf` command.
- Download the proof, check all references have DOIs, follow the links and check the references.
 - Whedon can help check references with the command `@whedon check references`
- Give the paper a proof-read and ask authors to fix typos.
- Ask the author to make a Zenodo archive, and report the DOI in the review thread.
- Check the Zenodo deposit has the correct metadata (title and author list), and request the author edit it if it doesn't match the paper.

- Run `@whedon set <doi> as archive.`
- Run `@whedon set <v1.x.x> as version` if the version was updated.
- Ping the `@openjournals/joss-eics` team on the review thread letting them know the paper is ready to be accepted.

At that point, the EiC/AEiC will take over to publish the paper.

It's also a good idea to ask the authors to check the proof. We've had a few papers request a post-publication change of author list, for example—this requires a manual download/compile/deposit cycle and should be a rare event.

3.5.4 Processing of rOpenSci-reviewed and accepted submissions

If a paper has already been reviewed and accepted by rOpenSci, the streamlined JOSS review process is:

- Assign yourself as editor and reviewer
- Add a comment in the pre-review issue pointing to the rOpenSci review
- Start the review issue
- Add a comment in the review issue pointing to the rOpenSci review
- Compile the paper and check it looks ok
- Tick off all the review checkboxes
- Go to to the source code repo and grab the Zenodo DOI
- Accept the paper

3.5.5 Sample letter to invite reviewers

```
Dear Dr. Jekyll,

I found you following links from the page of The Super Project and/or on Twitter. This
message is to ask if you can help us out with a submission to JOSS (The Journal of
↪Open
Source Software), where I'm an editor.

JOSS publishes articles about open source research software. The submission I'd like
↪you
to review is titled: "great software name here"

and the submission repository is at: https://github.com/< ... >

JOSS is a free, open-source, community driven and developer-friendly online journal
(no publisher is seeking to raise revenue from the volunteer labor of researchers!).

The review process at JOSS is unique: it is open and author-reviewer-editor
↪conversations
are encouraged.

JOSS reviews involve downloading and installing the software, and inspecting the
↪repository
and submitted paper for key elements. See https://joss.readthedocs.io/en/latest/
↪review_criteria.html
```

(continues on next page)

(continued from previous page)

```

Editors and reviewers post comments on the Review issue, and authors respond to the
↳ comments
and improve their submission until acceptance (or withdrawal, if they feel unable to
satisfy the review).

```

```

Would you be able to review this submission for JOSS? If not, can you recommend
someone from your team to help out?

```

```

Kind regards,

```

```

JOSS Editor.

```

3.5.6 Overview of editorial process

Step 1: An author submits a paper.

The author can choose to select an preferred editor based on the information available in our biographies. This can be changed later.

Step 2: If you are selected as an editor you get @-mentioned in the pre-review issue.

This doesn't mean that you're the editor, just that you've been suggested by the author.

Step 3: Once you are the editor, find the link to the code repository in the pre-review issue

Step 4: The editor looks at the software submitted and checks to see if:

- There's a general description of the software
- The software is within scope as research software
- It has an OSI-approved license

Step 5: The editor responds to the author saying that things look in line (or not) and will search for reviewer

Step 6: The editor finds ≥ 2 reviewers

- Use the list of reviewers: type the command `@whedon list reviewers` or look at list of reviewers in a [Google spreadsheet](#)
- If people are in the review list, the editor can @-mention them on the issue to see if they will review: e.g. `@person1 @person2 can you review this submission for JOSS?`
- Or solicit reviewers outside the list. Send an email to people describing what JOSS is and asking if they would be interested in reviewing.

Step 7: Editor tells Whedon to assign the reviewer to the paper

- Use `@whedon assign @reviewer as reviewer`
- To add a second reviewer use `@whedon add @reviewer2 as reviewer`

Note: The `assign` command clobbers all reviewer assignments. If you want to add an additional reviewer use the `add` command.

Step 8: Create the actual review issue

- Use `@whedon start review`

- An issue is created with the review checklist, one per reviewer, e.g. <https://github.com/openjournals/joss-reviews/issues/717>

Step 9: Close the pre-review issue

Step 10: The actual JOSS review

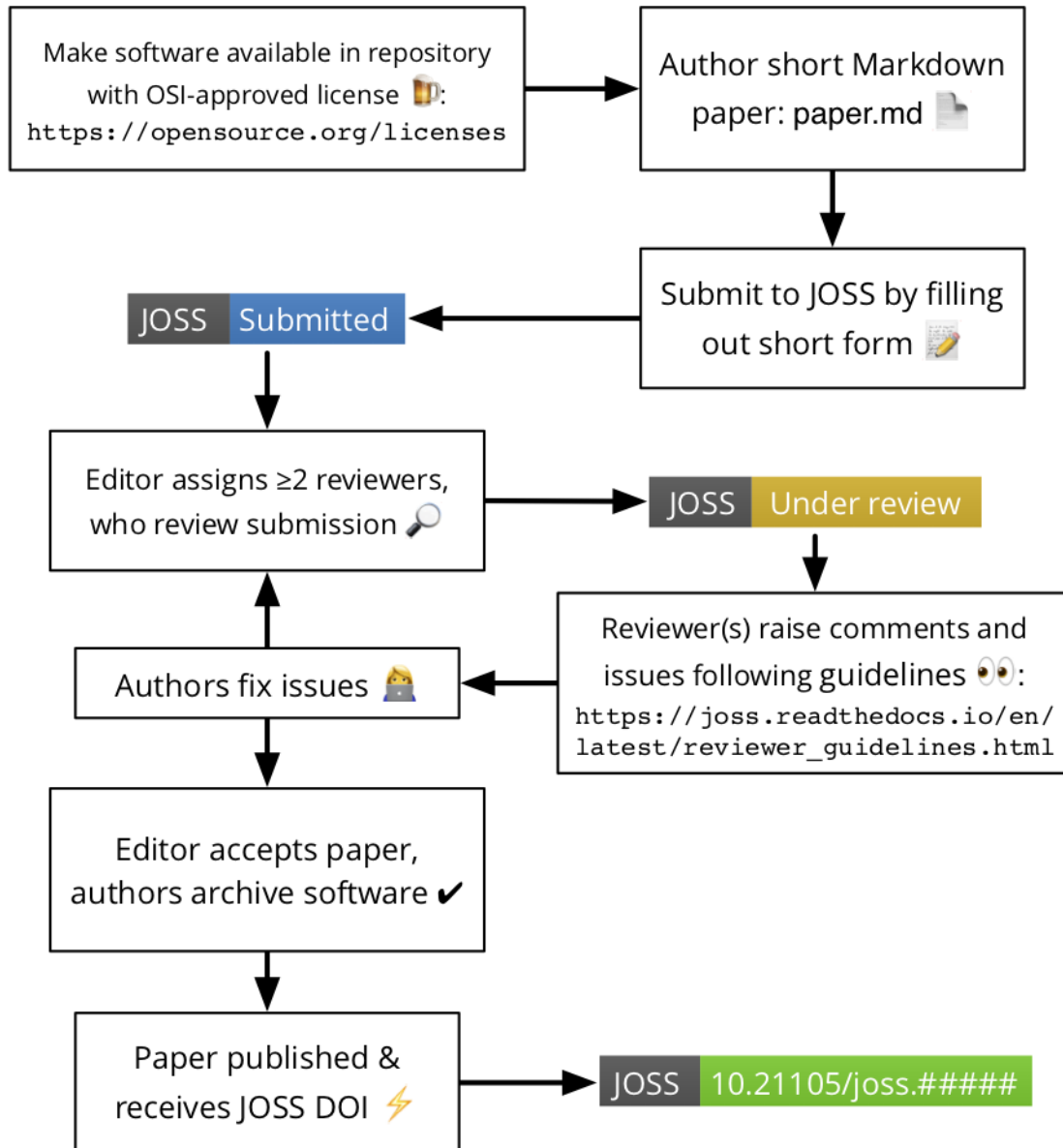
- The reviewer reviews the paper and has a conversation with the author. The editor lurks on this conversation and comes in if needed for questions (or CoC issues).
- The reviewer potentially asks for changes and the author makes changes. Everyone agrees it's ready.

Step 11: The editor pings the EiC team to get the paper published

- To get the paper published, ping the `@openjournals/joss-eics` team on the review thread letting them know the paper is ready to be accepted.

Step 12: Celebrate publication! Tweet! Thank reviewers! Say thank you on issue.

3.5.7 Visualization of editorial flow



flow

Editorial

3.5.8 Expectations on JOSS editors

3.5.8.1 Responding to editorial assignments

As documented above, usually, papers will be assigned to you by one of the AEiCs. We ask that editors do their best to respond in a timely fashion (~ 3 working days) to invites to edit a new submission.

3.5.8.2 Continued attention to assigned submissions

As an editor, part of your role is to ensure that submissions you're responsible for are progressing smoothly through the editorial process. This means that once or twice per week we ask that you check your GitHub notifications and/or your editorial dashboard (e.g. <http://joss.theoj.org/dashboard/youreeditorname>) for updates to the papers you are handling.

If reviews go stale

Sometimes reviews go quiet, either because a reviewer has failed to complete their review or an author has been slow to respond to a reviewer's feedback. **As the editor, we need you to prompt the author/or reviewer(s) to revisit the submission if there has been no response within 7-10 days unless there's a clear statement in the review thread that says an action is coming at a slightly later time, perhaps because a reviewer committed to a review by a certain date, or an author is making changes and says they will be done by a certain date.**

Whedon has [functionality](#) to remind an author or review to return to a review at a certain point in the future. For example:

```
@whedon remind @reviewer in five days
```

3.5.9 Out of office

Sometimes we need time away from our editing duties at JOSS. The [joss-reviews](#) repository has the [OoO bot](#) installed which means you can mark yourself as out of the office (and unable to respond to reviews) for a period of time e.g.:

Mark yourself as OoO in one of the reviews you're editing in the [joss-reviews](#) repository like this:

```
/ooo January 18 until February 2
```

Ooo bot will then respond to any mentions in the [joss-reviews](#) repository to let people know you're away.

Note, if you're planning on being out of the office for more than two weeks, please let the JOSS editorial team know.

3.6 Interacting with Whedon

Whedon or @whedon on GitHub, is our editorial bot that interacts with authors, reviewers, and editors on reviews.

@whedon can do a bunch of different things. If you want to ask @whedon what it can do, simply type the following in a review or pre-review issue:

```
@whedon commands
```

This will return an output something like this (the exact response depends upon whether or not you're an editor):

```
# List all of Whedon's capabilities
@whedon commands

# Assign a GitHub user as the sole reviewer of this submission
@whedon assign @username as reviewer

# Add a GitHub user to the reviewers of this submission
@whedon add @username as reviewer

# Remove a GitHub user from the reviewers of this submission
```

(continues on next page)

(continued from previous page)

```

@whedon remove @username as reviewer

# List of editor GitHub usernames
@whedon list editors

# List of reviewers together with programming language preferences and domain_
↳expertise
@whedon list reviewers

# Change editorial assignment
@whedon assign @username as editor

# Set the software archive DOI at the top of the issue e.g.
@whedon set 10.0000/zenodo.00000 as archive

# Set the software version at the top of the issue e.g.
@whedon set v1.0.1 as version

# Open the review issue
@whedon start review

EDITORIAL TASKS

# Compile the paper
@whedon generate pdf

# Compile the paper from alternative branch
@whedon generate pdf from branch custom-branch-name

# Remind an author or reviewer to return to a review after a
# certain period of time (supported units days and weeks)
@whedon remind @reviewer in 2 weeks

# Ask Whedon to accept the paper and deposit with Crossref
@whedon accept

# Ask Whedon to check the references for missing DOIs
@whedon check references

```

3.6.1 Author commands

A subset of the Whedon commands are available to authors (and reviewers):

3.6.1.1 Compiling papers

When a pre-review or review issue is opened, @whedon will try to compile the JOSS paper by looking for a `paper.md` file in the repository specified when the paper was submitted.

If it can't find the `paper.md` file it will say as much in the review issue. If it can't compile the paper (i.e., there's some kind of Pandoc error), it will try and report that error back in the thread too.

Note: If you want to see what command @whedon is running when compiling the paper, take a look at the code [here](#).

Anyone can ask @whedon to compile the paper again (e.g., after a change has been made). To do this simply comment on the review thread as follows:

```
@whedon generate pdf
```

3.6.1.1 Compiling papers from a non-default branch

By default, Whedon will look for papers in the default git branch. If you want to compile a paper from a non-default branch, this can be done as follows:

```
@whedon generate pdf from branch custom-branch-name
```

3.6.1.2 Finding reviewers

Sometimes submitting authors suggest people they think might be appropriate to review their submission. If you want the link to the current list of reviewers, type the following in the review thread:

```
@whedon list reviewers
```

3.6.2 Editorial commands

Most of @whedon's functionality can only be used by the journal editors.

3.6.2.1 Assigning an editor

Editors can either assign themselves or other editors as the editor of a submission as follows:

```
@whedon assign @editorname as editor
```

3.6.2.2 Adding and removing reviewers

Reviewers should be assigned by using the following commands:

```
# Assign a GitHub user as the sole reviewer of this submission
@whedon assign @username as reviewer

# Add a GitHub user to the reviewers of this submission
@whedon add @username as reviewer

# Remove a GitHub user from the reviewers of this submission
@whedon remove @username as reviewer
```

Note: The `assign` command clobbers all reviewer assignments. If you want to add an additional reviewer use the `add` command.

3.6.2.3 Starting the review

Once the reviewer(s) and editor have been assigned in the `pre-review` issue, the editor starts the review with:

```
@whedon start review
```

Important: If a reviewer recants their commitment or is unresponsive, editors can remove them with the command `@whedon remove @username as reviewer`. You can also add new reviewers in the `REVIEW` issue, but in this case, you need to manually add a review checklist for them by editing the issue body.

3.6.2.4 Reminding reviewers and authors

Whedon can remind authors and reviewers after a specified amount of time to return to the review issue. Reminders can only be set by editors, and only for `REVIEW` issues. For example:

```
# Remind the reviewer in two weeks to return to the review
@whedon remind @reviewer in two weeks
```

```
# Remind the reviewer in five days to return to the review
@whedon remind @reviewer in five days
```

```
# Remind the author in two weeks to return to the review
@whedon remind @author in two weeks
```

Note: Most units of times are understood by Whedon, e.g., *hour/hours/day/days/week/weeks*.

Important: For reviewers, the reminder will only be triggered if the reviewer's review is outstanding (i.e., outstanding checkboxes).

3.6.2.5 Setting the software archive

When a submission is accepted, we ask that the authors create an archive (on [Zenodo](#), [figshare](#), or other) and post the archive DOI in the `REVIEW` issue. The editor should add the `accepted` label on the issue and ask `@whedon` to add the archive to the issue as follows:

```
@whedon set 10.0000/zenodo.00000 as archive
```

3.6.2.6 Changing the software version

Sometimes the version of the software changes as a consequence of the review process. To update the version of the software do the following:

```
@whedon set v1.0.1 as version
```

3.6.3 Accepting a paper (dry run)

Whedon can accept a paper from the review issue. This includes generating the final paper PDF, Crossref metadata, and depositing this metadata with the Crossref API.

Topic editors can ask for the final proofs to be created by Whedon with the following command:

```
@whedon accept
```

On issuing this command, Whedon will also check the references of the paper for any missing DOIs. This command can be triggered separately:

3.6.3.1 Check references

```
@whedon check references
```

Note: Whedon can verify that DOIs resolve, but cannot verify that the DOI associated with a paper is actually correct. In addition, DOI suggestions from Whedon are just that—i.e., they may not be correct.

3.6.4 Accepting a paper (for real)

If everything looks good with the draft proofs from the `@whedon accept` command, the (associate) editors-in-chief can take the additional step of actually accepting the paper with the following command:

```
@whedon accept deposit=true
```

Note: This command is only available to the editor-in-chief or associate editors-in-chief.

3.7 Installing the JOSE application

To be written...

For now, please take a look at the [JOSS codebase](#).