# JOSS Documentation

Open Journals

Feb 10, 2019

# Author and Reviewer Guides

The Journal of Open Source Software (JOSS) is a developer friendly journal for research software packages.

JOSS is academic journal (ISSN 2475-9066) with a formal peer review process that is designed to *improve the quality of the software submitted*. Upon acceptance into JOSS, a CrossRef DOI is created and we list your paper on the JOSS website.

CHAPTER 1

# About this site

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

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

- Our announcement blog post describing some of the motivations for starting a new journal
- This paper in Computing in Science and Engineering introducing JOSS
- This paper in PeerJ CS describing the first year of JOSS
- The about page on the main JOSS site

CHAPTER 2

# Submitting a paper to JOSS

If you'd like to submit a paper to JOSS, please take a look at the author submission guidelines in the *Submitting a paper to JOSS* section.

CHAPTER 3

# Sponsors and affiliates



JOSS 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 JOSS) is a NumFOCUS-sponsored project.

## 3.1 Submitting a paper to JOSS

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.

### 3.1.1 Typical paper submission flow

Before you submit we need you to:

- Make software available in an open repository (GitHub, Bitbucket etc.) and include an OSI approved open source license
- Make sure that the software complies with the JOSS review criteria.
- Author a short Markdown paper `paper.md` with a title, summary, author names, affiliations, and key references. See an example here.

- (Optional) create a metadata file and include it in your repository describing your software. This script automates the generation of this metadata.

### 3.1.2 What should my paper contain?

JOSS welcomes submissions from broadly diverse research areas. For this reason, we request that authors include in the paper some sentences that would explain the software functionality and domain of use to a non-specialist reader. Your submission should probably be somewhere between 250-1000 words.

In addition, your paper should include:

- A list of the authors of the software and their affiliations
- A summary describing the high-level functionality and purpose of the software for a diverse, *non-specialist audience*
- A clear statement of need that illustrates the purpose of the software
- A list of key references including a link to the software archive
- Mentions (if applicable) of any ongoing research projects using the software or recent scholarly publications enabled by it

As this short list shows, JOSS papers are only permitted to contain a limited set of metadata (see header below), Statement of Need, Summary, and Reference sections. You can see an example accepted paper here. Given this paper format, a "full length" paper is not permitted, e.g., software documentation such as API (Application Programming Interface) functionality should not be in the paper and instead should be outlined in the software documentation.

---

**Important:** You paper will be reviewed by one or more reviewers in a public GitHub issue. Take a look at the review criteria to better understand how your submission will be reviewed.

---

### 3.1.3 Example paper and bibliography

This example `paper.md` is adapted from *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 2
    orcid: 0000-0000-0000-0000
    affiliation: 2
affiliations:
 - name: Lyman Spitzer, Jr. Fellow, Princeton University
   index: 1
 - name: Institution 2
```

(continues on next page)

(continued from previous page)

```
   index: 2
date: 13 August 2017
bibliography: paper.bib
---

# 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. The source code for ``Gala`` has been
archived to Zenodo with the linked DOI: [@zenodo]

# Mathematics

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

Double dollars make self-standing equations:

$$\Theta(x) = \left\{\begin{array}{l}
0\textrm{ if } x < 0\cr
1\textrm{ else}
\end{array}\right.$$


# Citations

Citations to entries in paper.bib should be in
[rMarkdown](http://rmarkdown.rstudio.com/authoring_bibliographies_and_citations.html)
format.

# 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{zenodo,
    Abstractnote = {Gala is a Python package for Galactic astronomy and gravitational␣
↪dynamics. The bulk of the package centers around implementations of gravitational␣
↪potentials, numerical integration, and nonlinear dynamics.},
    Author = {Adrian Price-Whelan and Brigitta Sipocz and Syrtis Major and Semyeong␣
↪Oh},
    Date-Modified = {2017-08-13 14:14:18 +0000},
    Doi = {10.5281/zenodo.833339},
    Month = {Jul},
    Publisher = {Zenodo},
    Title = {adrn/gala: v0.2.1},
    Year = {2017},
    Bdsk-Url-1 = {http://dx.doi.org/10.5281/zenodo.833339}
}

@article{gaia,
    author = {{Gaia Collaboration}},
    title = "{The Gaia mission}",
    journal = {\aap},
    archivePrefix = "arXiv",
```

(continues on next page)

(continued from previous page)

```
    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},
    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 when they are mentioned in the paper body.

### 3.1.4 Submitting your paper

Submission then is as simple as:

- Filling in the short submission form
- Waiting for reviewers to be assigned over in the JOSS reviews repository: https://github.com/openjournals/joss-reviews

### 3.1.5 Submission requirements

- The software should be open source as per the OSI definition
- The software should have an **obvious** research application
- You should be a major contributor to the software you are submitting
- The software should be a significant contribution to the available open source software that either enables some new research challenges to be addressed or makes addressing research challenges significantly better (e.g., faster, easier, simpler)
- The software should be feature complete (no half-baked solutions) and designed for maintainable extension (not one-off modifications). Minor 'utility' packages, including 'thin' API clients, are not acceptable.

In addition, the software associated with your submission must:

- Be stored in a repository that can be cloned without registration

- Be stored in a repository that is browsable online without registration
- Have an issue tracker that is readable without registration
- Permit individuals to create issues/file tickets against your repository

JOSS publishes articles about research software. This definition includes software that: solves complex modeling problems in a scientific context (physics, mathematics, biology, medicine, social science, neuroscience, engineering); supports the functioning of research instruments or the execution of research experiments; extracts knowledge from large data sets; offers a mathematical library, or similar.

### 3.1.6 Submission fees

There are no fees for submitting or publishing in JOSS. You can read more about our cost and sustainability model here.

#### 3.1.6.1 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 JOSS 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.

#### 3.1.6.2 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 submission 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.

### 3.1.7 The review process

After submission:

- One or more JOSS reviewers are assigned and the review is carried out in the reviews repository
- Authors respond to reviewer-raised issues (if any are raised) on the submitted repository's issue tracker. Reviewer 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, issue a DOI for the archive, and update the review issue thread with your DOI
- After assignment of a DOI, your paper metadata is deposited in CrossRef and listed on the JOSS website
- And that's it

If you want to learn more about what the review process looks like in detail, take a look at the reviewer guidelines.

## 3.2 The JOSS review process

Firstly, thank you so much for agreeing to review for the Journal of Open Source Software (JOSS), we're delighted to have your help. This document is designed to outline our editorial guidelines and help you understand our requirements for accepting a submission into the JOSS. Our review process is based on a tried-and-tested approach of the rOpenSci collaboration.

### 3.2.1 Guiding principles

We like to think of JOSS as a 'developer friendly' journal. That is, if the submitting authors have followed best practices (have documentation, tests, continuous integration, and a license) then their review should be rapid.

For those submissions that don't quite meet the bar, please try to give clear feedback on how authors could improve their submission. A key goal of JOSS is to raise the quality of research software generally and you (the experienced reviewer) are well placed to give this feedback.

A JOSS review involves checking submissions against a checklist of essential software features and details in the submitted paper.

We encourage reviewers to file issues against the submitted repository's issue tracker. **When you have completed your review, please leave a comment in the review issue saying so.**

You can include in your review links to any new issues that you the reviewer believe to be impeding the acceptance of the repository. (Similarly, if the submitted repository is a GitHub repository, mentioning the review issue URL in the submitted repository's issue tracker will create a mention in the review issue's history.)

## 3.3 Review criteria

### 3.3.1 The JOSS paper

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

- A list of the authors of the software
- Author affiliations
- A summary describing the high-level functionality and purpose of the software for a diverse, non-specialist audience
- A clear statement of need that illustrates the purpose of the software
- Mentions (if applicable) of any ongoing research projects using the software or recent scholarly publications enabled by it
- A list of key references including a link to the software archive

---

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

---

### 3.3.2 Review items

#### 3.3.2.1 Software license

There should be an OSI approved license included in the repository. Common licenses such as those listed on choosealicense.com are preferred. Note there should be an actual license file present in the repository not just a reference to the license.

> **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 Documentation

There should be sufficient documentation for you, the reviewer to understand the core functionality of the software under review. A high-level overview of this documentation should be included in a `README` file (or equivalent). There should be:

##### 3.3.2.2.1 A statement of need

The authors should clearly state what problems the software is designed to solve and who the target audience is.

##### 3.3.2.2.2 Installation instructions

There should be a clearly-stated list of dependencies. Ideally these should be handled with an automated package management solution.

> **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

##### 3.3.2.2.3 Example usage

The authors should include examples of how to use the software (ideally to solve real-world analysis problems).

##### 3.3.2.2.4 API documentation

Reviewers should check that the software API is documented to a suitable level.

> **Good:** All functions/methods are documented including example inputs and outputs **OK:** Core API functionality is documented **Bad (not acceptable):** API is undocumented

---

**Note:** The decision on API documentation is left largely to the discretion of the reviewer and their experience of evaluating the software.

---

##### 3.3.2.2.5 Community guidelines

There should be clear guidelines for third-parties wishing to:

- Contribute to the software

- Report issues or problems with the software
- Seek support

#### 3.3.2.3 Functionality

Reviewers are expected to install the software they are reviewing and to verify the core functionality of the software.

#### 3.3.2.4 Tests

Authors are strongly encouraged to include an automated test suite covering the core functionality of their software.

> **Good:** An automated test suite hooked up to an external service such as Travis-CI or similar **OK:** Documented manual steps that can be followed to objectively check the expected functionality of the software (e.g. a sample input file to assert behaviour) **Bad (not acceptable):** No way for you the reviewer to objectively assess whether the software works

### 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 JOSS 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 An important note about 'novel' software

Submissions that implement solutions already solved in other software packages are accepted into JOSS provided that they meet the criteria listed above and cite prior similar work.

#### 3.3.3.3 What happens if the software I'm reviewing doesn't meet the JOSS 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're more than happy to give the author as long as they need to make these modifications/improvements.

#### 3.3.3.4 What about submissions that rely upon proprietary languages/development environments?

As outlined in our author guidelines, submissions that rely upon a proprietary/closed source language or development environment are acceptable provided that they meet the other submission requirements and that you, the reviewer, are able to install the software & verify the functionality of the submission as required by our reviewer guidelines.

If an open source or free variant of the programming language exists, feel free to encourage the submitting author to consider making their software compatible with the open source/free variant.

## 3.4 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.

---

### 3.4.1 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`

---

#### 3.4.1.1 Finding reviewers

At this point, the handling editor's job is to identify reviewer(s). We like to have two reviewers per submission. If the editor is comfortable with their own assessment of the submission, one reviewer may be sufficient. If the editor feels particularly unsure of the submission, maybe a third reviewer can be recruited.

To recruit reviewers, the handling editor can ping candidates on Twitter, or mention them in the `PRE-REVIEW` issue with their GitHub handle. 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.

---

#### 3.4.1.2 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.

### 3.4.2 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.

### 3.4.3 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 add the `accepted` label on the issue, 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.
- Add the "accepted" label.
- 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.

### 3.4.4 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

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.
```

### 3.4.5 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 >= 1 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 @reviwer2 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**

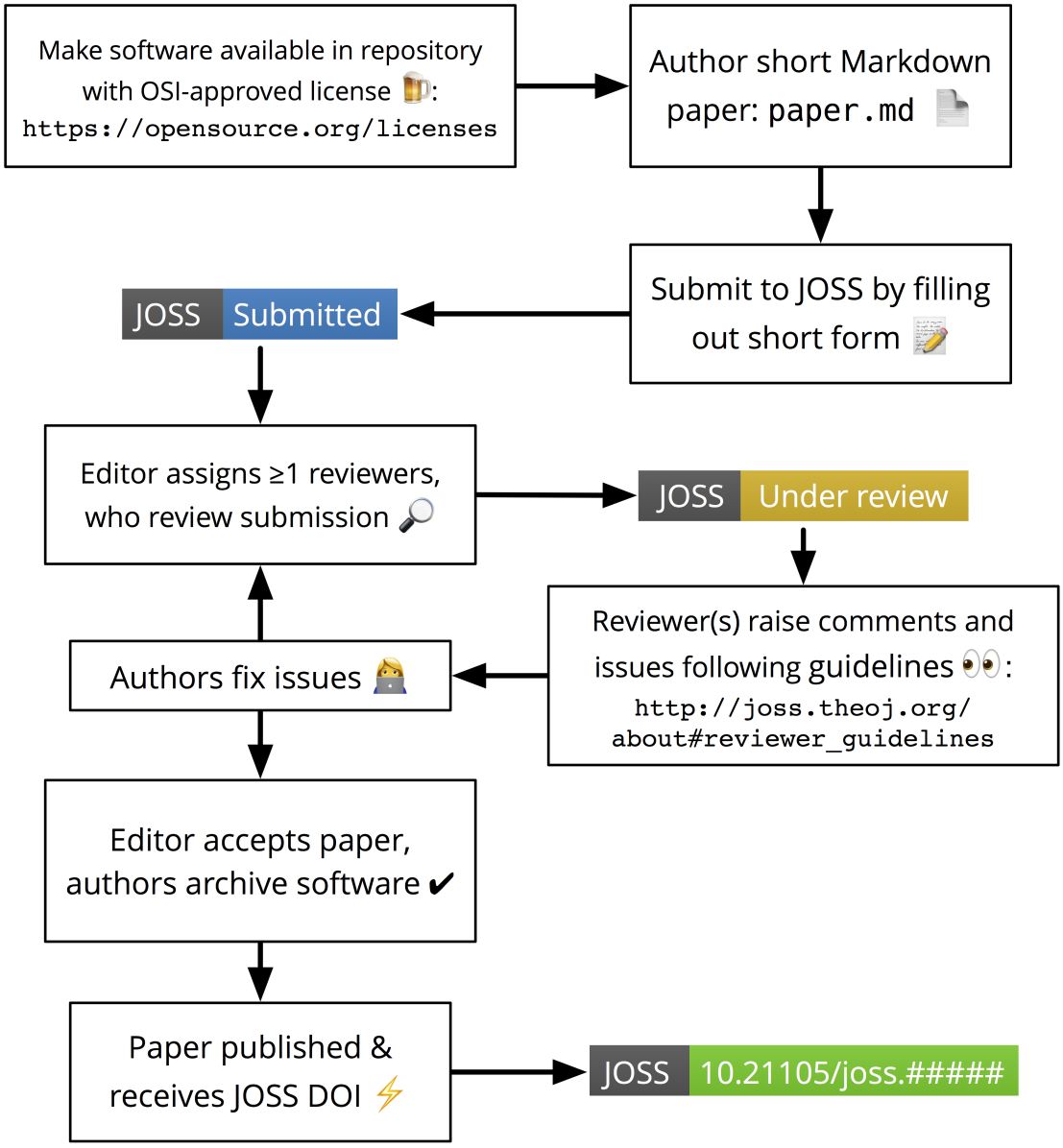
- The reviewer reviews the paper and has a conversation with the author. The editor lurks on this conversation and comes in if needed for questions (or CoC issues).
- The reviewer potentially asks for changes and the author makes changes. Everyone agrees it's ready.

**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.**

### 3.4.6 Visualization of editorial flow




### 3.4.7 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.**

## 3.5 Interacting with Whedon

Whedon or `@whedon` on GitHub, is our editorial bot that interacts with authors, reviewers, and editors on JOSS 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 JOSS `review` or `pre-review` issue:

```
@whedon commands
```

This will return an output something like this:

```
# 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
@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

# Open the review issue
@whedon start review

   Experimental Whedon features

# Compile the paper
@whedon generate pdf
```

### 3.5.1 Author commands

#### 3.5.1.1 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 JOSS 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
```

#### 3.5.1.2 Finding reviewers

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

```
@whedon list reviewers
```

### 3.5.2 Editorial commands

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

#### 3.5.2.1 Assigning an editor

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

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

#### 3.5.2.2 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.

#### 3.5.2.3 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.

#### 3.5.2.4 Assigning 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 <archive doi> as archive
```

## 3.6 Installing the JOSS application

To be written...

For now, please take a look at the JOSS codebase.