Sustaining Community-Driven Software for Astronomy in the 2020s

Erik Tollerud <etollerud@stsci.edu>, Arfon Smith, Adrian Price-Whelan, Kelle Cruz, Dara Norman, Gautham Narayan, Stuart Mumford, Alice Allen, Chi-kwan Chan, Brian Cherinka, Alex Drlica-Wagner, Dan Foreman-Mackey, Adam Ginsburg, Andre Gradvohl, Joseph Harrington, David Hogg, Jeyhan Kartaltepe, Jamie Kinney, Nirav Merchant, Ivelina Momcheva, Nicholas Murphy, Joshua Peek, Molly S. Peeples, Timothy Pickering, David Rodriguez, Lior Shamir, Manodeep Sinha, Brigitta Sipőcz, Jennifer Sobeck, Megan Sosey, Heloise Stevance, Peter Teuben, Dany Vohl, Benjamin Weiner

Endorsed by:
Thomas Aldcroft, Alice Allen, Mehmet Alpaslan, Lauren Anderson, Geert Barentsen, Dino Bektesevic, Jose Benavides, Bruce Berriman, Michael Blanton, James Bosch, Daina Bouquin, Larry Bradley, Greg Bryan, Douglas Burke, Keaton Burns, Derek Buzasi, Juan B. Cabral, José Vinícius de Miranda Cardoso, Boquan Chen, Will Clarkson, Michelle Collins, Lia Corrales, Matthew Craig, Steven Crawford, Shawn Domagal-Goldman, Chuanfei Dong, Meredith Durbin, Jacqueline K Faherty, Will Farr, Luigi Forschini, V. Zach Golkhou, Hans Moritz Günther, Heiko Hafok, ChangHoon Hahn, Nimish Hathi, Christina Hedges, Song Huang, Cameron Hummels, Cameron Hummels, Emily Hunt, Daniela Huppenkothen, Stephanie Juneau, Marten van Kerkwijk, Wolfgang Kerzendorf, Iva Laginja, Casey Law, Jerome de Leon, Ting Li, Pey Lian Lim, Alex I. Malz, Yao-Yuan Mao, Peter Melchior, Bruno Merin, Bryan Miller, Maryam Modjaz, Timothy Morton, Susan Mullally, Ricardo Ogando, John K. Parejko, Dante Paz, Sarah Pearson, Klaus Pontoppidan, Benjamin Pope, David Rapetti, Meredith Rawls, Justin Read, Thomas Robitaille, Gregory Rudnick, Swapnil Sharma, Sanjib Sharma, David Shupe, Joshua Speagle, Tjitske Starkenburg, Federico Stasyszyn, Ole Streicher, Grant Tremblay, Francisco Villaescusa-Navarro, Johanna M Vos, Benjamin A. Weaver, Amanda Weltman, Andrew Wetzel, Peter K. G. Williams, Benjamin Winkel
Summary

Software is critical to astronomical research. Sharing and sustaining astronomical software has long-term impacts on scientific outcomes. However, support for this has been uneven, creating significant risks. Thus, we highlight changes that will enable a sustainable software sharing system for astronomy and astrophysics in the next decade.

Recommendations

1. Existing grant programs should fund astronomy software development.

Software that enables astronomy discovery should be allowable as a sole or primary deliverable for existing programs funding astronomy research (e.g., NSF AAG, CAREER, NASA ROSES, postdoctoral prize fellowships), as long as the software is of use to the scientific community (or has sufficient potential to do so), and/or has intellectual merit and impact. Calls for proposals should be amended to reflect this by adding language to emphasize software contributions. For example, “The Program also considers proposals for projects, tools, and software that enable or enhance astronomical research.”

2. Agencies and the astronomical community should fund and value sustaining of astronomical “community software infrastructure” projects.

Funding agencies and the community as a whole should support funding of domain-specific community-developed software projects, e.g., The Astropy Project, SunPy, rOpenSci. The agencies should recognize such projects as vital infrastructure, equal in importance to physical facilities such as national observatories, and create grant programs to fund domain-specific and interdisciplinary community-development efforts such as workshops and training programs in addition to code development.

While this could be part of existing grant programs, a dedicated program to support community software development and maintenance should be considered to ensure the longevity of the program and software. For example, we estimate that a new program to fund ~2 large community projects and ~6 smaller software projects (e.g., from individual PIs) per cycle would cost ~$4 million per year, significantly cheaper than other critical infrastructure such as national facility maintenance.
3. Agencies and institutions must include cultivating a sustainable research software ecosystem as part of their review criteria.

As part of their review criteria for all astronomy grant programs, funding agencies should: (1) Require a plan for how software will be managed (and sustained long-term when relevant) to support the science of the grant, (2) Require a description of how proposed software development fits into and supports the wider ecosystem of available tools, and (3) Prefer (although not require) programs that develop community software as part of their funded activities. In general, a grant contributing code to open source community software projects would meet these goals. The goals should be communicated to grant review panelists and applicants and also be considered by the broader astronomy science community (both when on panels and wider contexts).

Why software matters, today and in the next decade

Software is critically important and valuable in the present Petabyte era of discovery in astronomy: Well-organized, maintained software enhances the utility, accessibility, and reliability of scientific investigation and results. In this White Paper, we emphasize these points in the context of shared software; that is, standalone analysis scripts written and not shared are not the subject of this discussion. However, when such scripts are shared with collaborators, made available online, or contributed to a larger project, this makes the software part of a community process, and hence the astronomy community is the target of this discussion. The discussion here broadly applies to both “open” (publicly-available, sometimes open source or open development, e.g., The Astropy Project) and “closed” (e.g., software shared within a collaboration) shared software.

Regardless of the type of software community, we note also that a key part of sustaining software communities is in fostering a diverse and inclusive environment for users and developers; By their very nature as a social process, shared software efforts must include diversity and inclusion. This paper focuses on the software itself, however, so we refer the reader to recommendations in Norman et al. 2019 for more detail.

Software is everywhere

Software is an integral and growing part of the scientific endeavor: It is responsible for driving the control systems of instruments, the operation of surveys, the processing of raw data products, the extraction of physical parameters, and the theoretical modeling of physical systems. **Software is critical to modern science and is the foundation of computational astronomy.** This reality is well-recognized by the scientific community: Of nearly 2000 scientists surveyed, over 91% reported that scientific software is
important or very important for their research, and that it would be impractical for them
to carry out their research without it (Hanny et al. 2009). Moreover, the rapid increase in
the size and complexity of astronomical experiments and the data they produce has led
to an increasing demand for astronomical software development, from large facilities to
individual PIs. For example, the LSST project has allocated ~25% of the construction
budget ($187M) for data management software, infrastructure, and services¹, and 10%
of their computation resources to the community.

At the same time, the way astronomical software is developed has changed
rapidly, in large part driven by broader changes in the cultural norms of modern
software development and a shift towards open source software being the “new normal”
(LeClair 2016; Gnau 2017). Large experimental projects (such as LSST, JWST, DESI,
DKIST) are funding the development of extensive codebases and releasing these tools
as open source software²³⁴⁵. Individuals are also becoming increasingly likely to
distribute and share their codes broadly with the astronomical community and
mechanisms for publishing these software products have expanded as a result (AAS
Publishing 2015; GitHub 2016; Smith et al. 2018; Astronomy & Computing 2013; Allen
et al. 2013). In the present and future data- and computation-intensive reality, where
software permeates scientific investigation, it is critical that the contributions of software
developers are recognized and that individuals are provided with the necessary
resources to succeed (see Smith et al. 2019 for concrete recommendations).

Software encodes knowledge

As datasets grow larger and our analysis methods become more sophisticated, an
increasing fraction of the scholarly endeavor is expressed in software. This presents
both opportunities and challenges. One potential opportunity is that the “centralization”
of astronomy (i.e., the trend towards smaller numbers of large facilities, often with open
datasets) means that any software built (and shared) leveraging these facilities has a
higher reuse potential. A major potential risk, identified by others (Donoho et al. 2009;
Yale Roundtable Declaration 2010), is that as a large part of the scientific process
becomes encoded in software, if this software is not shared (e.g., as open source),
reviewed, or tested, the reproducibility of our science is increasingly at risk.

² http://github.com/lsst
³ http://github.com/spacetelescope
⁴ http://github.com/desihub
⁵ https://github.com/DKISTDC
Shared Software for scientific reproducibility

As scientific projects become increasingly large in scope, ever more distinct software components are required to analyze and produce the final research data products. These software components must be open to a community of researchers to verify and validate for the research to be reproducible - a core element of the scientific method. Unfortunately, much of the code being used is not documented, let alone complete with unit tests that can validate performance. Moreover, managing a multitude of dependencies creates a non-linear increase in the overall project complexity. Subtle undetected errors driven by these issues are surprisingly common in the computational science literature (Collberg & Proebsting 2016), and likely even more so in more fields where such methods are applied, like astronomy.

Hence, in modern astronomy research, "software reproducibility" is necessary for scientific reproducibility so the astronomy community should actively incorporate best practices to make our science reproducible. While progress has been made in developing technologies to improve this (e.g., easy and widely available software repositories like GitHub, containerization technologies like Docker, etc.), many of these technologies are primarily used by early adopters in science rather than the mainstream. By contrast, standardized shared software tautologically leads to more reproducible software, so supporting the community that maintains this software directly improves reproducibility. Thus our recommendations support both the software for reproducibility and individuals with expertise to lead the charge for the community.

Progress in the last decade

Many of the issues highlighted in this White Paper are not new. In particular, we note a white paper from the Astro2010 decadal review with a similar scope: Weiner et al. (2009). That paper discussed areas of concern and specific recommendations, some of which have improved materially, while others have seen little progress. We discuss the recommendations of that paper here to provide a historical context and guidance for the future.

1. Weiner et al. (2009): “create a[n] open central repository location at which authors can release software and documentation”. Enormous progress in this area has been achieved in the last decade. Open source software repositories, chief among them GitHub⁶, have become the defacto standard for storing software in astronomy. The wider adoption of Python has improved the packaging and release process due to the Python Package Index⁷ and the

---

⁶ https://github.com
⁷ https://pypi.python.org
ecosystem of easy-to-host documentation tools that support Python, like Sphinx\(^8\) and ReadTheDocs\(^9\). While these are not perfect solutions for some languages and science domains, the presence of a much larger and better-funded user base (open source industry software) has made them stable enough to be adopted for astronomy’s use and can likely continue to do so for the foreseeable future.

2. Weiner et al. (2009): “Software release should be an integral and funded part of astronomical projects”. Progress in this area has been mixed. While large efforts of this decade like LSST, JWST, DESI, or DKIST have large first-class software components, many smaller projects or individual grant-level efforts continue to treat maintainable or reproducible software as an afterthought rather than a necessary part of the scientific endeavour, often due to a lack of funding to support such efforts. While funding agencies like the NSF, DOE, and NASA have required data management plans, there has been less progress on establishing firm requirements or expectations of sustainable software (although a recent NASA-driven consideration of these issues is available in the National Academies of Sciences, Engineering, and Medicine 2018).

3. Weiner et al. (2009): “Software release should become an integral part of the publication process.” and “The barriers to publication of methods and descriptive papers should be lower.”. Considerable progress has been made in this area, but that is not the focus of this paper (see Smith et al. 2019).

4. Weiner et al. (2009): “Astronomical programming, statistics and data analysis should be an integral part of the curriculum” and “encourage interdisciplinary cooperation”. While some progress has been made in this area, many challenges remain. We defer further discussion of this to White Papers focused on this topic (e.g., Norman et al. 2019, Smith et al. 2019)

5. Weiner et al. (2009): “more opportunities to fund grass-roots software projects of use to the wider community”. While such projects have grown remarkably in the last decade (see the Open Development section below), major challenges still remain in sustainably funding them; this motivates our core recommendations.

6. Weiner et al. (2009): “institutional support for science programs that attract and support talented scientists who generate software for public release.”. Some of the elements of this recommendation have grown with the advent of “Big Data” and “Data Science” academic positions in astronomy. There has also been a growing recognition of the importance of research-oriented software positions, particularly in Europe (e.g. Research Software Engineers International 2018). However, there are very few viable pathways for researchers who develop

---

\(^{8}\) [http://www.sphinx-doc.org](http://www.sphinx-doc.org)  
\(^{9}\) [https://readthedocs.org/](https://readthedocs.org/)
software of broad use as part of their research program if it is not easily pitched in terms of current fashions in computational science. Hence this is also a key element of our recommendations.

There is one final distinction to be highlighted relative to the last decade: It is clear that software has become more mission-critical, and will continue to gain in importance and in complexity. These generic concerns about software development are therefore multiplied across the deeper layers of software, making all the issues more broadly applicable. The urgency in addressing these issues will grow in the coming decade.

Community software as a research multiplier

Collaboratively-developed community software has an increasingly large impact throughout the astronomy community. For example, the whole scientific software ecosystem in Python (the most popular language in astronomy - see Momcheva & Tollerud 2015) is built on community-developed software like NumPy (van der Walt et al. 2011), SciPy (Jones et al. 2001–), Matplotlib (Hunter 2007), or other parts of the so-called “NumFOCUS Stack”. More domain-specific projects such as Astropy project (Astropy Collaboration et al. 2013, 2018) and SunPy (SunPy Community et al. 2015) capture the expertise of a broad range of astronomers, and have a wealth of features that cannot be reproduced by solitary researchers.

While the mere existence of such software open for all to use is immediately apparent, there are several ancillary benefits to such community software efforts:

● The more the community participates, the more the project will reflect their specific needs and applications, even if it is built on a more general framework.
● The code is typically inspected by more people, so problems and their solutions will be found more quickly (Raymond 2001).
● There is usually more free energy and conscious effort for documentation and documentation-related tools, and a larger base to help support new users.
● It is easier to train scientists to help produce professional-quality software if a core of professional engineers supports them. Community projects provide a larger-scale social understanding of how that interaction can happen.
● The cost is shared and substantially lower than would be for independent re- implementations of common tools and functionality.
● These projects speed the cycle of science by providing useful implementations of common tasks, freeing researchers to focus on their specific science.
● When built as part of an underlying broader ecosystem, community software often gains the direct benefit of contributions “upstream”, e.g., improvements in
core math libraries made by computer scientists can flow down to astronomy without any direct effort in astronomy.

Together, these factors mean that the impact of code developed by a community is both multiplied by contributions from other sources to the same ecosystem and can reduce the time each individual researcher needs to spend on code development; It is thus a “research multiplier.” Note that the community-developed software need not strictly be open source, though the majority of these projects are: The benefits of community development extend to both open and closed source projects, the primary difference being that the potential size of an open project is by definition larger than a closed one, and most of the above points scale positively with community size.

Open development and open collaboration

While a large amount of software in Astronomy is now open source, and has been for decades, a major development in recent years has been the growth of open development. This form of collaborative software development accepts and, in many cases, depends wholly on contributions from the wider community to the software project. Development of the code and discussion around it is conducted in the open using industry-standard platforms like GitHub or GitLab. In most cases, policy discussions and decisions also occur in an open internet mailing list. The chief examples of projects like this in astronomy are The Astropy Project and SunPy.

This kind of open development model is not limited to astronomy projects. There are many examples of key, large-scale software projects that are developed in the open, the largest example of which is the Linux kernel. Developing software in this way introduces technical and sociological challenges, which have been met by tools such as git, online collaboration tools such as GitHub that enable workflows to scale to hundreds or thousands of contributors, and the hard work of organizers and code reviewers to set up and maintain a positive culture that enables contributions to continue.

These kinds of open collaborations enable many different stakeholders (both astronomer users and dedicated developers) to collaborate on a software project, often coming from a diverse\(^\text{10}\) set of perspectives. Such collaboration is much easier in open development projects because the developers are users (and vice versa), while in the closed case communication between these groups requires additional effort. The user-developer interaction built in when the community of users and developers overlap results in each contribution being more valuable, since individual contributions are used by a broader group. It also means more work can be done with less funding, because the efforts of individual contributors are pooled into a “neutral” space that can arbitrate

\(^{10}\) Diverse here has multiple meanings, but we highlight this as an area where the typical meaning of diversity & inclusion serves to directly improve the quality of software and the research it enables.
via the community process. Moreover, the open nature of the collaboration means that stakeholders have the ability to drive the direction and priorities of the project simply by contributing to it. Because many of these stakeholders are the users themselves, it also can serve to optimize the applicability-to-effort ratio.

Community software problems and solutions

There is a disconnect between the increasing importance of community software and the funding available for such projects. In particular, the future of many widely used projects, including Astropy and services such as astrometry.net, is uncertain. These major community projects are generally unfunded despite the vital role they play for astrophysics as a whole (for additional discussion, see Muna et al. 2016). While many feature “in-kind” contributions from user missions, such support depends on the vagaries of mission priorities rather than the needs of the community itself.

Hence, the benefits outlined above cannot be realized if such efforts are not supported by funding agencies, large missions, and indeed the astronomical community as a whole. There are presently few incentives to encourage community efforts: In some cases, such software development is either not allowed by a grant program or tacked on as an afterthought. (“My grad student can build that reduction pipeline on the way to their thesis.”) Where software grant programs do exist, they often focus on building specific applications into interdisciplinary tools (e.g., NSF CSSI and DIBBs), rather than applying general software to specific domains. They also, as a rule, do not emphasize community-building elements like contribution policy documents, documentation of user workflows, or community coordination. They also generally do not support ongoing maintenance of existing software. Our recommendations therefore focus on providing incentives and opportunities to support social software development by missions and individuals alike. This will be critical to keep up with the ever more software-rich future.

Software is alive

“This open source stuff is free. But it’s free like a puppy. It takes years of care and feeding.” - Hanselman 2006

The present funding model for academia supports work done over fixed timescales. In contrast, shared astronomical software often requires ongoing maintenance, updates to add or adapt functionality, and development of new features over timescales that can be longer than any individual researcher or developer’s involvement. Like any living thing, software will not survive without proper care and feeding, and must adapt to a continually changing environment.
The software stack is always changing. Our community must be adaptable. One challenge of the present scientific software stack is that it evolves with time, both internally (i.e., the code changes) and as new tools emerge (i.e., the dominant software language changes). These changes often bring new features or benefits, like making the code easier to use and faster, or taking advantage of other developments in the wider technological ecosystem. A recent example is Python replacing IDL and IRAF as the most popular programming language within astronomy (Momcheva & Tollerud 2015): Reasons for this change are likely the lack of license fees in Python, the extensive open-source ecosystem of Python libraries for scientific computing, and the broader usage of Python (which most notably means more training and marketable skills). This demonstrates the fact that change in the software stack is inevitable and to be encouraged when useful to the community.

However, the disruption caused by these changes can be harmful if not addressed as part of well-validated software for research. Hence there is always a cost-benefit analysis needed to address changing technologies and changing science environments that must be considered in order to yield the most benefit. Disruptions from evolving software can be ameliorated by continuing education programs for researchers at all levels. That is not the focus of this White Paper, but Norman et al. 2019 provide comprehensive recommendations for this topic.

Software needs to be sustainable

Any software that is meant to be used more than once requires maintenance. Data sets change (or grow to Petabyte scale), bugs are discovered, computer architectures change, dependencies evolve, and users change their understanding of the intent of the software. This circumstance leads to the concept of software sustainability: practices both within the software itself and of those who develop it that make it practical to maintain the software for an arbitrarily long time. For astronomy software to be sustainable (Katz et al. 2018; Wilson et al. 2014), it should:

1. Be both testable and tested (i.e., it is correct, and anyone can check that correctness).
2. Be readable and useable by multiple people, and can therefore evolve to fulfill its intent over time as development and scientific conditions change.
3. Have a viable pathway to be maintained past the original author.
4. Be able to respond to (likely-to-evolve) users’ needs.

That said, even software maintained by a third party organization (e.g., Harris Geospatial Solutions for IDL) is not guaranteed future permanent status within astronomy (Momcheva & Tollerud 2015). As astronomy shifts towards a more
community-developed, open source set of tools, the different constituents of the astronomy community must develop an understanding of the origin of this software and how they are a part of its development, maintenance, and long-term sustainability: **Software consumers (individual astronomers):** Most individual researchers make heavy use of software tools developed by their peers but do not routinely participate in the development or validation of the software; This is the norm and is acceptable. However, complete ignorance of the origin of software creates a risk to the sustainability of the projects and individuals responsible for creating and maintaining the software. For example, if the consumers do not realize the software they are using comes from other researchers, they may not support hiring, tenure, etc. of those who build that software, thereby stopping them from producing and maintaining the software itself. Even as software consumers, astronomers should be aware of the origin of the software they are using and realize that they have a critical role to play in the community by 1) providing feedback to software projects by providing bug reports, feature requests, feedback, and by contributing other resources like documentation if they have relevant expertise; 2) recognizing that software is created by people, and that supporting the work of their peers (financially, socially, or even emotionally) who spend time creating these tools is necessary for the tools they use to even exist; and 3) recognizing and advocating for the concept that using a shared set of tools can improve all of science for less cost. **Individual software creators (astronomers and engineers):** Creators of shared software are responsible for both setting the expectations of the community (e.g., be explicit about whether they intend to support users or encourage contributions) and for being aware of the community they are building for (e.g., typical user communities do not have as much technical expertise and therefore require help to both learn how to use the software and understand why it is useful). Sustainable software creation therefore comes with a responsibility to either perform that maintenance or to communicate to users their limitations. **Institutional software creators (projects/missions/facilities):** Observatories and missions (e.g., LSST, JWST, DKIST), especially in development and construction phases, spend significant resources developing software both for internal operations and for their community to analyze and interpret data products from their facilities. Institutional software creators need to be incentivized to upstream (i.e., contribute back innovations to community software packages) their software where possible, thereby contributing to the large ecosystem of software available to the general astronomy community. As discussed earlier, community software can be a research-multiplier, but for this to happen, large projects must recognize their role in the community software ecosystem and shift towards being active contributors rather than consumers/users.
References

- AAS Publishing. 2015, online
- Astronomy & Computing. 2013, online
- GitHub. 2016, online
- Gnau, S. 2017, online
- Hunter, J. D. 2007, Computing In Science & Engineering, 9, 90
- Jones, E., Oliphant, T., Peterson, P., et al. 2001–, SciPy: Open source scientific tools for Python
- LeClair, H. 2016, online
- Raymond, E. S. 2001, The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary (Sebastopol, CA, USA: O'Reilly & Associates, Inc.)
- Smith, A.M. 2019, Elevating the Role of Software as a Product of the Research Enterprise. To be submitted as Astro2020 SoP White Paper
- SunPy Community, T., Mumford, S. J., Christie, S., et al. 2015, Computational Science and Discovery, 8, 014009
- Varoquaux, G. 2013 online
- Yale Roundtable Declaration. 2010, Computing in Science & Engineering, 12, 8