Toward A Sustainable Software Development Model for Heliophysics

Authors: Will T. Barnes[1], James Juno[2], Jeffrey W. Reep[3], Jack Ireland[4], Paul J. Wright[5], Sarah A. Spitzer[6], B. L. Alterman[7], David Stansby[8], Emily Lichko[9]
Cosigners: Monica G. Bobra[5], Joel Dahlin[4,10], Aleida Higginson[4], Michael S. F. Kirk[4,11], Kevin Reardon[12], Yeimy Rivera[6], Viacheslav Sadykov[13], Sam Schonfeld[14], Ashley R. A. Smith[16], J. L. Verniero[15], Sam Wallace[4], Micah Weberg[1], Phyllis Whittlesey[15]
[1]NRC Postdoc residing at the Naval Research Laboratory, [2]University of Iowa, [3]Naval Research Laboratory, [4]NASA Goddard Space Flight Center, [5]W. W. Hansen Experimental Physics Laboratory, Stanford University, [6]University of Michigan, [7]Southwest Research Institute, [8]Mullard Space Science Laboratory, University College London, [9]University of Arizona, [10]Universities Space Research Association, [11]Atmospheric and Space Technology Research Associates, [12]National Solar Observatory, [13]Bay Area Environmental Research Institute, [14]Institute for Scientific Research, Boston College, [15]Space Sciences Laboratory, University of California Berkeley, [16]School of GeoSciences, University of Edinburgh

Statement of Problem

Software is a critical component of heliophysics research. We define "software" as cohesive libraries or codebases that provide functionality useful in many research workflows[1], not smaller pieces of code produced by individual researchers for specific tasks or use cases. For decades, most heliophysics software has been developed for highly specialized individual tasks. As a result, most software is often poorly documented and lacks comprehensive tests. Moreover, many scientists end up recreating software that others have already developed. In cases where heliophysics software is properly developed, it is usually done so by early-career researchers (ECRs) who are funded in an *ad hoc* way as part of proposals for which the software is a secondary consideration. This has led to a fractured software ecosystem in which rapid publication is prioritized at the expense of software that enables repeatable and well-documented science. This is not sustainable.

Piecemeal software development negatively impacts heliophysics in multiple ways. (1) The absence of software versioning is a barrier to reproducibility. Past studies often cannot be meaningfully reproduced as the state of the software at the time of publication is unknown. (2) In the absence of thorough unit and regression tests, critical software flaws are likely to go unnoticed, potentially undermining published scientific results. (3) Deemphasizing software-focused heliophysics careers impedes the retention of ECRs who are producing and contributing to research software. (4) Inadequately funding and supporting research software development effectively siloes the developers and software. As a result, heliophysics research software often lacks interoperability within and between subdomains and software must be duplicated for each use, thus wasting research effort and dollars.

To this end, we set forth several goals that the solar and heliophysics community should achieve by 2050:

1. Software development is actively recognized and rewarded through a dedicated funding

[1] E.g. the IDL software for prepping SDO/AIA images distributed in SolarSoftware or the sunpy Python package.

mechanism that enables career stability.
2. Research software for heliophysics funded by NASA is open source, openly-developed, and adheres to established standards for versioning, documentation, and testing.
3. The software ecosystem for heliophysics is sustainable and interoperable.

Below, we outline several short-term goals, to be accomplished within the next 10 years, and long-term goals, to be accomplished within the next 20 years, that support our vision of software development for heliophysics in 2050.

Short-Term Goals: the next 10 years

NASA should provide concrete guidance to researchers regarding best practices in citing software. Unlike research publications, the use of software is often not rewarded through citations, and researchers who developed this software are not proportionally rewarded for their effort. In a survey of software usage in the solar and heliophysics community, Bobra et al. (2020, Sol. Phys., 295 57) found that 73% of respondents said they have cited software in their research, but only 42% said they did so routinely. When asked why they did not cite software in their research, 53% responded that they were not sure how to properly cite software. To this end, we recommend that NASA provide concrete guidance to researchers both on how to cite software as well as how to make their software citable[2]. Appendix B of Bobra et al. (2020) provides concrete recommendations for citing software in research publications.

NASA should establish a funding line with the singular goal of supporting software development. In this funding model, proposals should address community software needs (e.g. modernizing a legacy, but popular MHD code, or developing a user-friendly scripting layer for an extensively used low-level piece of software). Any work funded under such a solicitation should meet the heliophysics community's software standards for documentation, testing, and interoperability[3]. Rather than having to make the case that a piece of software will answer an unknown question in heliophysics, competitive proposals need only demonstrate that the proposed development fills a significant community need and thus accelerates the pace of scientific discovery[4]. Such a funding program should also encourage collaboration between groups receiving these funds in order to reduce duplication of effort and foster an interoperable software ecosystem[5]. As an example, two instrument teams, both of whom maintain Python tools for analyzing spectroscopic data, could collaborate to develop a common set of code for dealing with spectral data and then individually develop tools specific to their instruments on top of this common codebase.

Early career researchers hired to develop mission-critical software should be supported for longer than typical postdoctoral tenures. If software development is required for a mission's success, money from the mission must be allocated to provide stability for that software development. ECRs hired to do this work should not be hired as temporary

---

[2] For example, researchers can obtain a DOI for software by uploading it to Zenodo or having the software itself reviewed by a research software journal like the Journal of Open Source Software (JOSS)
[3] These standards could be modeled after those established by the Python in Heliophysics community
[4] E.g., PlasmaPy's successful NSF Cyberinfrastructure for Sustained Scientific Innovation proposal
[5] A prototype of such a funding program is the recently-revamped NASA Heliophysics Data Environment Emphasis ROSES call that supports development of Python packages for heliophysics.

postdoctoral researchers, but as research scientists for the proposed lifetime of the mission. These extended positions (~5-7 years) will allow research scientists to produce high-quality, mission-critical software without having to balance the typical demands of research publications and proposal writing. Such stability would also create opportunities for a different academic career track with advancement driven by software development and its corresponding science.

Long-Term Goals: the next 20 years

NASA should create a centralized Software Center. Such a center would provide a career path for interdisciplinary research in software engineering and could be part of an existing NASA facility or hosted at a separate institution. The principal goal of such a facility would be to provide a career path for scientists performing research relevant to the stated science goals of NASA who also want to dedicate significant effort to the improvement and maintenance of research software. Within this center, software engineering best practices (e.g. unit and regression tests, thorough documentation) would be both expected and rewarded through promotion and, eventually, career stability. Furthermore, this Software Center could have separate divisions for different types of research software. As an example, a modeling division[6] could maintain and improve PDE solvers and large scale 3D MHD codes of interest to the whole community, while a data division could maintain and improve data reduction and analysis software. Such a division could principally focus on the maintenance and development of the SunPy core library and the associated ecosystem of packages, similar to the manner in which the Space Telescope Science Center provides support for the development of the Astropy Python package.

Programs should be created to train future generations to write well-documented, tested, and maintainable code. Such programs would manifest as summer programs for undergraduates at the Software Center. Additionally, an extended-stay graduate student program could explicitly train, through apprenticeship, students in software engineering. Such a program is critically important as graduate programs in physics and astronomy do not typically include such training, and it is not reasonable to expect students to wade through a generic core computer science curriculum at their home institutions. This would standardize the training that is useful and appropriate for software development in heliophysics. This is critical to promote a culture of open development in which software is developed for and by the community. Training programs should also be targeted at ECRs interested in pursuing research full-time, helping scientists be more invested in the code they use. In this vision, a number of full time researchers would take active leadership roles and promote best development practices within the community.

[6] This would partly take inspiration from the NASA Community Coordinated Modeling Center (CCMC) which hosts and executes, on request, a number of large-scale space science and space weather models. The CCMC also funds the development of analysis tools for output from these models (e.g., Kamodo)