DOI: 10.1002/pra2.281

LONG PAPERS

WILEY

Finite and infinite games: An ethnography of institutional logics in research software sustainability

Nicholas Weber

Information School, University of Washington, Seattle, Washington, USA

Correspondence
Nicholas Weber, Information School, University of Washington, Seattle, WA.
Email: nmweber@uw.edu

Abstract

Modern research is inescapably digital, with data and publications most often created, analyzed, and stored electronically, using tools and methods expressed in software. While some of this software is general-purpose office software, a great deal of it is developed specifically for research, often by researchers themselves. Research software is essential to progress in science, engineering, and all other fields, but it is often not developed, shared, or stored in a sustainable way. The following paper presents findings from an ethnography of two research software projects that have, over the last 10 years, cooperatively organized development efforts to produce important software enabling scientific breakthroughs in both astronomy and macromolecular modeling. The work of these two projects are framed in terms of James Carse's model of *finite* and *infinite* games. I argue that by incentivizing institutional governance that resembles the design of an infinite game, funding agencies can increase the sustainability of research software and improve various aspects of data-driven scientific discovery.

1 | INTRODUCTION

Common tasks solved by research software often include the generation, analysis, visualization, and processing of data. These software solutions are, just as often, generalizable beyond the immediate needs of an individual researcher, research group, or a particular research effort. However, there are few institutional and personal incentives to develop generalizable research software, package this software for reuse, create meaningful documentation, and share software in open repositories. Each of these activities requires substantial investments of time, money, and effort. For researchers the return on this investment may be minimal - software is rarely cited in scholarly literature (Howison & Bullard, 2016; Hwang, Fish, Soito, Smith, & Kellogg, 2017; Hsu et al. 2019; Park & Wolfram, 2019, tenure and promotion committees rarely consider software contributions (Moher et al., 2018), and grant funding often acknowledges software development as a byproduct of rather than a substantive contribution to a research project (Broman et al., 2017; Siepel, 2019).

These challenges are despite increasing evidence of the value of software sharing and reuse in addressing research challenges that require fast and efficient community response. For example, the Medical Research Center in the UK is currently using a 13 year old pandemic simulation codebase to model control measures for COVID-19. In doing so, they have attracted collaborators from Microsoft, the Abdul Latif Jameel Institute for Disease and Emergency Analytics, and the WHO Collaborating Centre for Infectious Disease Modelling to document, refactor, and extend this code. While the original author of the simulation code acknowledged its numerous imperfections,[1] the ability to start from an existing

83rd Annual Meeting of the Association for Information Science & Technology October 25-29, 2020.

working model saved time, money, and effort in combating a global pandemic.

Contemporary research is littered with similar examples - imperfect software that is valuable for one purpose can be made more broadly useful by being shared, properly documented, and made available for sustainable reuse. Finding ways to encourage and support research software so that it remains accessible for future improvements and uses is a primary goal of federal funding agencies, like the National Science Foundation.

2 | SOFTWARE SUSTAINABILITY IN THE USA

Between 1998-2016 the US National Science Foundation (NSF) made more than 18,000 awards related to the development of research software. This diverse investment portfolio, totaling over $9.6 billion dollars in federal funding, includes projects that have produced new software libraries, programming environments, visualization tools, and software services (Ram et al., 2018). Alongside advances in other cyberinfrastructure components, such as data repositories and high-performance computing centers, research software now enables advanced scientific discovery and analytical innovations that are transformative to many domains of knowledge production (Katz and Ramnath, 2015).

This profound investment of financial capital into research software has also introduced new challenges for a sustainable scientific enterprise: Students need to be trained to effectively use advanced software (Wilson, 2014), research results are difficult to reproduce without democratized access to underlying hardware and software sources (Stodden and Miguez, 2014), and academic career advancement, predicated mainly on contributions in peer-reviewed conference and journal publications, often does not acknowledge nor give credit for the work of developing, maintaining, or sharing research software (Howison, 2015; Weber and Thomer, 2015). In short, the last two decades of scientific funding has ushered in paradigmatic changes to research and development activities, and has significantly expanded the software dependencies necessary for advancing experimentation and discovery.

The Software Infrastructure for Sustained Innovation (SI2) program at NSF recognized and made important progress on many of the research software challenges noted above (Katz and Ramnath, 2015). However, the sponsors for that program also recognized that one of the most vexing problems, one that cuts across all of these issues, is how - given the time-limited, and finite financial commitments available to funding agencies like the NSF - can valuable research projects sustain the human and technical software infrastructures necessary to advance scientific understanding?

In 2018, the US Research Software Sustainability Institute (URSSI) was funded to investigate and support sustainable software development by NSF. URSSI has, over the last two years, held a number of community-listening workshops and a widely-distributed survey that engaged important stakeholder communities of scientific software in the US (Carver, Gesing, Katz, Ram, & Weber, 2018). Each of these activities have been designed to help NSF learn about the software produced and used by the US scientific and engineering community, and the ways that NSF stakeholders go about practically sustaining this research ecosystem.

As part of URSSI's work, the lead author has conducted a year long ethnogic study of sustainable software development. The goals of this ethnographic work are to help URSSI better generate domain-specific examples of sustainable software practices, and to inform the design of a general institute aimed at improving software development for research. The sections that follow present a brief summary of the design of an ethnographic study of two long-term research software projects that have sustained their work through unique governance, and software development practices. Following this section, we present findings from this ethnography and frame the results in terms of institutional design.

3 | INSTITUTIONAL ETHNOGRAPHY & RESEARCH DESIGN

Ethnographers of institutions are often interested in the lived experience of people working together to build and maintain social relations (Campbell et al., 2006). In the context of this paper, an institution is the set of material practices and symbolic systems including assumptions, values, and beliefs that individuals (or groups of individuals) use to provide meaning to routine activity. These activities can range from simple naming and group identity, to complex tasks related to organizing collective action, and reproducing or replicating patterns of collective action over time (Thornton & Ocasio, 2008). Through observation (in-person and virtual), interviews, and archival document analysis ethnographers can provide descriptive causal interpretations about why some institutions succeed and others fail (Rankin, 2017).

To better understand the institutional arrangements that lead to sustainable research software, we designed a comparative ethnography of two projects: *Astropy* - a python software project used broadly in Astronomy research; and, *Rosetta Commons* - a platform for

macromolecular modelling. Both projects have been successful at producing high-quality research software, and maintaining infrastructure that supports this software over the last decade. We therefore sought to compare how these projects successfully accomplished these sustainability activities, and generalize from the similarities and differences in their work. The formal research questions that we seek to answer are related to social relations, shared artifacts, and governance:

- How are *social relations* initiated and maintained through *shared artifacts* (code) in research software development?
- How do *shared artifacts* structure or get structured by the *institutional governance* of a software project?; and
- How do forms of i*nstitutional governance* get sustained as they adjust to external and internal pressures of maintaining *social relations*?

These research questions reflect our assumption that social relations, shared artifacts, and governance are all highly related to sustainability in a research software ecosystem. We therefore seek to understand and differentiate how - through collective action processes - these concepts impacted the practical, everyday work of two different research software projects.

To operationalize this comparative study we first created a brief historical account of the way that each project formed. Through these histories we also sought to explain how each project's organizational structure (e.g., roles, funding, etc) has evolved over time. After creating these historical explanations, we purposefully recruited participants from each project to read and evaluate the narrative that we had created. This reading and editing of a historical account was a purposeful intervention by our research team - we were attempting to induce reflexivity in each interviewee, and gain an understanding of how social relations, shared artifacts, and governance were articulated.

In total, we interviewed 23 users, developers / contributors, and project leaders in Astropy and Rosetta Commons. A questionnaire was developed in advance and shared with interview participants. Participants ranged in seniority from graduate students to tenured professors. Each interview lasted approximately 40 minutes, and took a semi-structured form. Interviews were audio-recorded and then transcribed. The lead author on this paper then developed a codebook related to social relations, shared artifacts, and governance, and coded each interview thematically (Vaismoradi, Jones, Turunen, & Snelgrove, 2016).

In addition to interviews the lead author also collected the following data: Observations from a series of software workshops in Astronomy (2018), and an annual retreat for the software in biomolecular modeling (2019). Observations took the form of field notes that were also transcribed, and coded thematically. We also collected digital trace data of organizational activities, such as email forum posts, Github issue discussions, and we joined Slack and Gitter groups (with invitation from administrators) for both projects. Using these resources we then turned to the small, but influential literature on research software engineering and peer production to contextualize and further refine themes that cut across both projects. We developed external and internal validity of our observational data in the following ways: We shared interpretations of events and historical readings with participants at each stage of analysis. We also sought feedback on our initial coding of interview data, and asked participants for their help in evaluating synthesis of these observations with the existing literature on research software sustainability.

The remainder of this paper is structured as follows: We first present a brief historical narrative of both Astropy and Rosetta Commons. We then compare general features of each project in a summary table. Next, we discuss emergent themes that were observed and emerged from interviews with participants. We then turn to an explanation of sustainable software institutions in terms of finite and infinite games drawn from the work of James Carse.

4 | CASE STUDIES OF SUSTAINABLE RESEARCH SOFTWARE

4.1 | Astronomy - Astropy

In the late 1990s the astronomy community began to move away from using Interactive Data Language (IDL) in favor of free open-source alternatives. This move was precipitated by an increasingly sophisticated research and development ecosystem in Astronomy that required the broad sharing of software for interpreting telescope observations that had increased substantially in size and complexity (Boscoe, 2019). By the mid 2000s, the programming language Python was being used extensively, and astronomy specific-packages were being developed in a number of USA and European labs working with large-scale astronomy data. The adoption of Python in astronomy also had to do in part with a desire for an open alternative to IDL, and the paradigm of object-oriented programming (as opposed to vector-oriented) becoming pervasive in scientific computing.

The adoption of Python led to important innovations in not just astronomy software, but general libraries for

data processing and analysis. For example, 'numarray' developed by astronomers in 2003 directly led to 'NumPy', and 'plotting' developed in 2007 by John Hunter led to 'matplotlib'. Both of these packages, NumPy and matplotlib, are used extensively in scientific computing. However, in astronomy there was also much redundancy in development efforts with Python. Prior to "social coding" infrastructures like Github there was no mechanism for discovering software other than personal networks of collaborators. This led to duplicated work and a failure for the astronomy community to converge on shared software projects that could harness collective efforts under a single core library. Muna et al. describe the situation in the early-2000s as not just uncoordinated, but also stifling "...it would be one thing if there were widespread disagreement on the next direction of software development, but given that the path forward was clear and almost universally agreed upon, it would have been reasonable to expect that some institution – one that had led in software development previously or somewhere new – would have taken on the task of writing the next generation of tools, in Python, that astronomers would be needing." (2016). Two major community-based efforts have since emerged in the post-IDL era of Astronomy: yt and Astropy.

Astropy, formed in 2011, is self-described as a community based project to "develop a single core package for Astronomy in Python and foster interoperability between Python astronomy packages." (2018) The original project documentation further describes the vision for the project as "...to avoid duplication for common core tasks, and to provide a robust framework upon which to build more complex tools...This vision is not set in stone, and we are committed to adapting it to whatever process and guidelines work in practice." (2011). Astropy is licensed under the three-clause BSD license which is a share-alike license that allows for reuse and redistribution in binary form without any liability for copyright holders. Astropy participants in this study noted that, true to the original vision, the focus of the project continues to be on providing generalized interfaces, including a set of well documented APIs that allow new developers to access, use, and eventually contribute to Astropy in meaningful ways. Astropy is now used by the Association of Universities for Research in Astronomy (AURA); the Hubble Space Telescope and James Webb Space Telescope - among other influential astronomy research laboratories.

Practically, astropy includes a core set of packages that provide for common astronomical computing tasks, such as data wrangling, modeling, visualization, statistical computing, numerical computing, text and image processing, as well as a growing set of subject area libraries that extend these capabilities. Participants in this study stressed repeatedly a difference of Astropy from other previous software contributions in astronomy - it is community developed and user focused rather than guided by the specific needs of a single survey or laboratory. Community focus means that as missions or surveys develop specific imaging or pipelines for data processing Astropy can be extended to satisfy general use cases, or specific libraries can be contributed for specific tasks.

In 2014, after 3 years of general community financial support, Astropy transitioned to fiscal sponsorship by NumFOCUS (a non-profit organization that provides for administrative services to open-source software projects). Although broadly used, well documented, and highly coordinated the project has continued to face dilemmas in attracting the level of funding that is necessary for sustainability. A provocative paper, written by a community of users called this the Astropy Problem (Muna et al., 2016) which echoes the free-rider problem in commons governance. In short, it describes community developed software that is extensively used and that depends exclusively on volunteer labor for its upkeep and maintenance. Without direct financial support from the numerous surveys and institutions that depend upon this infrastructure the project faces extensive coordination and sustainability issues. Muna's paper proposes a sponsorship model, and licensing fee which would help cover the costs of producing and maintaining Astropy, but neither effort has been, as of yet, taken up.

4.2 | Molecular modeling - Rosetta commons

Macromolecular modeling in the field of biophysics and bioengineering includes the analysis and prediction of molecular structures and activities. The field has advanced rapidly over the last two decades, due in part to the development of a common suite of modeling software called Rosetta. The development of the software happened from 1997-2000 in David Baker's Lab at University of Washington in response to a need for a "structure prediction tool." (Baker et al., 2001). Rosetta was originally written in Fortran, but was later rewritten in C++ (v 2.0), and eventually a set of Python wrappers were written to make the code easier to interact with (v 3.0). The move to an object-oriented language, similar to Astropy, was motivated by the desire to increase learning opportunities of undergraduate and graduate researchers, and to generally encourage the development of new and extensible libraries that could be coordinated in a growing number of contributing macromolecular bioinformatics labs.

Rosetta has been extended considerably since its origin in a single university lab - it now helps to solve "common computational macromolecular" problems throughout

bioinformatics (e.g., industry, government labs, non-profits). Rosetta also has multiple functional modules that include RosettaAbinitio, RosettaDesign, RosettaDock, RosettaAntibody, RosettaFragments, RosettaNMR, RosettaDNA, RosettaRNA, RosettaLigand, RosettaSymmetry PyRosetta. RosettaScripts, ROSIE. Portions of the Rosetta framework were used to produce, and analyze data from the game FoldIT - which gained international attention for its novel use of crowdsourcing (Khatib et al., 2011).

Rosetta became The Rosetta Commons in 2003 - a non-profit entity that manages the intellectual property of the project. This move was necessitated by the authors of Rosetta, all of whom were part of the Baker Lab at UW, ending their postdocs and starting their own labs at new institutions. This group of postdocs wanted the ability to continue working on the codebase, sharing portions of development responsibilities amongst new postdocs, and building the basic framework of Rosetta into a suite of infrastructure tools. Rosetta is licensed by the University of Washington, and is free for academic use. Licenses for commercial use of Rosetta costs upwards of $40,000.

Rosetta has a developer and contributor program, but does not allow for open contributions. Developers must come from one of the 23 affiliated Rosetta Commons labs (which currently includes about 150 contributors to the core Rosetta library). The project is governed by a board that is elected from member labs, but runs mostly on a volunteer basis. A recurring grant from NIH has provided for Rosetta Infrastructure to be maintained, at a rate of approximately ~$500 K annually, since 2005. Over the last decade Rosetta Commons affiliate labs have won over 50 grants to build out complimentary algorithms, features, and extensions of Rosetta. The grants are mostly in support of basic research for student and postdoc developers - who also happen to be the most common software developers in macromolecular modeling. The project sustains itself practically by the member labs assuming responsibility for one or more components of the core architecture, and four dedicated infrastructure developers (full time employees) at the University of North Carolina.

4.3 | Astropy & Rosetta at a glance

The following Table 1 provides a point of comparison for the two projects based on some basic project demographics.

TABLE 1

	Astropy	Rosetta Commons
Founded	2011	2003
Fiscal Model	Donation, Grants	Grants, Licensing
License	3-clause BSD	Custom (Rosetta License)
Userbase (estimated size)	2500 forks (imperfect proxy)	10000 current licenses in use
Contributions	Open	Closed - member labs and approved secondary developers only.
Maintenance	Volunteer - (~7 core, and ~ 25 sub-package maintenance coordinators)	Core Infra team (4) paid to work on maintenance.
Coordination	Weekly telecon, Gitter, Github Issues	RosettaCon (annual), telecon monthly, Slack, Github Issues (closed)

5 | OBSERVATIONS

In the following sections we offer some points of comparison for how the two projects handle issues of sustainability. Specific anecdotes or quotes are attributed to a participant [P] of each project- Astropy [A] or Rosetta [R]- and are numbered to provide for participant anonymity.

5.1 | Contributors and Maintainers

Throughout our interviews we asked questions that could explain the success of each project in attracting new contributors and sustaining knowledgeable maintainers. Notably, both projects have scaled from a small group of early committed individuals to a distributed development model of contributors working and coordinating tasks across the world. Our questions on growth focused on how new and existing contributors were initiated, and how newcomers were helped to learn, understand, and meet the expectations for contributing to a project.

Astropy was conceived of as having a distributed coordination structure, and so its early documentation sketched out a method of receiving and accepting new contributions. The initial Astropy leaders had planned initially to develop worked examples of how to make a code contribution [PA-2]. Over time this documentation has evolved into examples for not only how contributions are expected to be structured, but the exact workflow that new contributors can follow. One participant explained the strength of the documentation reflecting the "survey and mission" driven nature of Astronomy data production [PA-1]. Astropy currently has a thorough set of

contributor guidelines including a worked example for how to contribute via their git workflow (for beginners), and even a condensed example for making new contributions to Astropy. These guidelines are developed and maintained by designated individuals who are responsible for documentation upkeep - a task that could only be assigned once a proper governance structure was in place around the project [PA-2]. Making broader changes to Astropy's core code is formally facilitated through an Astropy Proposal for Enhancement (APE) process that follows the same model of proposals as the Python language. The APE is used to both coordinate new subpackage development as well as code reorganization, and other community wide proposals. Informally, the community discusses issues, proposals, training, and coordinates through a mailing list that is open for anyone to join. (Note: a further discussion of maintenance and retention is also addressed in the final section of this paper).

Rosetta Commons follows a very different contributor model due to its licensing scheme – which necessarily provides barriers to who can provide new code, propose new directions for the project, and even communicate with core developers. Both the process of contributing, as well as finding out who can contribute new code or fix bugs can be opaque. When asked to point to documentation about this procedure Rosetta participants had no clear answer other than private Github wikis. One commenter on the Rosetta Commons forums explained this to an external user as follows, "The Github account is just the repository of the source code - while you don't get the full history of the code and access to the cutting-edge version, when you download the Rosetta release, you only get a snapshot of that repository." Rosetta repository snapshots have no documentation about how to fix a bug, how to contribute an extension, or make a pull request to the repository. A participant explained the lack of openness as being less exclusionary and more about control of the project by member labs, "there certainly isn't a secret page which lays out the structure of the Rosetta library clearly" [PR-1]. Even the 23 contributing labs lack documentation for proposing broad changes or coordinating code clean-up. This was seen by many participants code clean-up and bug resolution as a problem that the infrastructure team was supposed to solve.

Much of the contributions of code come from just two sources: Member labs or a core Infrastructure team. There are 23 member labs of the Rosetta commons that collectively employ over 200 students. Many of these students and postdocs are responsible for maintaining and responding to bugs that are found in one or more of the sub-packages of Rosetta. Students are trained at individual labs about how to make pull requests to core and sub-packages of Rosetta. There were, as of participant reporting, no formal contributor guidelines that guide this training. Most participants described a Rosetta bootcamp at an annual event, RosettaCon, as their only formal training for using and making contributions to the project.

Besides students and postdocs, there is a team of core infrastructure developers and maintainers (four FTE) based at the University of North Carolina. The infrastructure team receives an annual NIH award for their salaries (continuing since 2005). These individuals do almost all servicing of commercial licensing holders and are responsible for the Rosetta core maintenance and new development. Students regularly interact with infrastructure team members, and each of the current infrastructure members are former Rosetta lab members. Infrastructure team members receive training on infrastructural elements of Rosetta hosting, provisioning, and deployment by UNC IT, but are not trained on the Rosetta software itself other than their experience as researchers. In interviewing students, we asked if becoming an infrastructure team member was seen as a coveted or admirable position – and were enthusiastically rebuffed. The individuals playing this role are well respected, but Rosetta students and postdocs do not view the positions as viable career paths.

6 | INTERFACES (PARTIALLY) EXPLAIN SUCCESS

Interviews with participants from both projects often steered towards questions of how and why their work had succeeded where others had failed, and what explains their collective success in sustaining a project over time. Participants from both projects described early decisions to privilege the development of well documented APIs, and attributed these decisions to their success.

The core contribution of Rosetta Commons to macromolecular modeling is a common interface that allows for protein prediction (and a host of other modeling tasks) to be generalized. APIs are designed and maintained only by the core infrastructure team of Rosetta.

The APIs developed for this purpose have matured alongside the growth of the project. One participant explained Rosetta was first commercially licensed in 2004 the project received an influx of funds – and decided to invest both their time and money in building out a suite of APIs that would make accessing new libraries much easier [PR-6]. The participant attributed this to the further commercial success of Rosetta, and to the capabilities of the APIs to allow for broad use of the modeling components of Rosetta.

Astropy participants also repeatedly described APIs as the key to success of the project over other Python projects in astronomy - notably work from the Large Synoptic Survey Telescope (LSST) in the mid-2000s that failed to be extended beyond that particular survey [PA-3]. Development and maintenance of API's in Astropy are the responsibility of sub-package and core maintainers. The maintainers that I talked to noted that this was one of their most important responsibilities, as one explained the dilemma in receiving credit for this work, "No one cites data and no cites software – I can't even imagine someone citing an API which is really some combination of... data access and software.. I guess it's the key [to Astropy] but I'm not sure anyone even knows I worked on it." [PA-3].

For project contributors, computational interfaces in both Rosetta and Astropy were seen as key to the early and long-term successes. Interestingly, research participants in both projects noted that they did not really consider such an interface to be a software contribution, and were unsure who developed or maintained an API. This invisibility of infrastructural middleware is a common theme in CSCW (e.g., Bietz, Paine, & Lee, 2013), but we argue in the conclusion that APIs require further attention from researchers attempting to understand contemporary software sustainability.

7 | ACKNOWLEDGEMENT OF USE

In interviews with both project developers and researchers we sought to understand motivations for working on shared software. While motivations for working on open-source projects have been thoroughly examined in other settings (Hars, 2002), we believed that asking questions about this behavioral aspect of research software sustainability could lend insights into how and why social relations were maintained through shared artifacts. The academic reward system, dependent upon citations in peer-reviewed literature, was rarely mentioned by either group as a motivation for contributing to software development. Instead, both Rosetta and Astropy developers described a sense of "duty" for maintaining research infrastructure that was important to their field. One participant succinctly put this as follows:

I'm good in my field, but I'm not going to make breakthroughs. That requires a lot of access that frankly I just do not have. I gave up on that in undergrad. But, I have made a number of important pull requests for Astropy that feel better to me than any publication I've ever been on. I mean, when people describe their data processing pipeline at a conference and I see something I've built - I'm like 'I did that! Your science is possible because of me' [PA-5].

When asked about citation and formal acknowledgement within publications many participants described key papers that were known as "trademarks" [PA-5] for a software contribution to either Rosetta and Astropy. Below, we further investigate the papers that were mentioned by participants, and we describe their reception by participants in interviews and observations.

The default citation for Rosetta is a 2004 paper explaining the capabilities for using the software in protein structure prediction (Rohl, Strauss, Misura, & Baker, 2004). The paper has received ∼1400 citations, and continues to receive ∼100 per year, which is far less than the use of Rosetta suggests based on active licenses. Participants acknowledged that much of the software development activities of students and postdocs goes unrecognized and unacknowledged. Participants from Rosetta provided conflicting thoughts about whether the lack of acknowledgement in software development was important. Students felt that this was their responsibility for being a member of a lab, and were overall enthusiastic about the opportunity to engage and interact with other labs via software development. One participant noted that after a year developing for Rosetta he began working hard to get pull requests accepted and to make contributions to public repositories so that he could get more credit for his contributions. I talked with only two PIs of Rosetta labs, but both saw the acknowledgement of software contributions as problematic for postdocs, and they noted many graduate students having to spend more and more time learning software engineering tasks and not being rewarded for this work. They also noted that undergraduate students increasingly viewed participation in their labs as a pathway to industry.

By comparison, Astropy's default publication (Robitaille et al., 2013) has received just 292 citations. This publication is promoted on the Astropy website, and documentation throughout the project encourages citation for the sake of acknowledgement. Participants in the Astropy project all stressed the major dilemma that this caused for attracting funding, demonstrating impact, and receiving credit. Community members have been active in software citation efforts in scholarly communications, and have encouraged sub-package developers and maintainers to pursue software specific publications [PA-1]. Muna et al, have gone a step further in their 2016 publication by trying to estimate the economic impact of Astropy. Using David A. Wheeler's (Wheeler, 2004)"SLOCCount" they estimate the cost of reproducing Astropy to be ∼$8.5 million, and the annual economic impact on astronomy alone to be ∼$1.5 million (2016). When asked whether or not this exercise was effective for funding participants had mixed feelings - One claimed that the figure was used in every conversation she had with funding agencies and is widely

cited in the astronomy software community, and another said she believes the figure did harm in trying to loosely put a "hedonic value" on something that could not be easily replaced [PA-6,7].

8 | MANAGEMENT, GOVERNANCE, AND RETENTION

Scholars of peer production and social computing have long been concerned with how newcomers are retained in distributed digital projects. Halfaker, Kittur, and Riedl (2011) and TeBlunthuis, Shaw, and Hill (2018) have found that quality control systems often, "limit the growth of peer production communities by deterring new contributors and that norms tend to become entrenched over time." This latter claim, that norms are entrenched over time, is by no means surprising, but has been shown to lead to poor sustainability as projects scale beyond small groups (n = 10) of participants. For example, Shaw and Hill show that founders of a wiki often claim more and more control as the diversity of participants (contributors) grows - which stifles newcomers and causes wikis to fail over time (2015). Similar findings in open source software studies show that transitioning newcomers from the periphery positions to a core set of developers and maintainers is one of the hardest tasks for distributed projects, but also is important to their long-term success (Crowston et al., 2007). We sought to understand how norms had evolved in the face of growth, and the ways that both Astropy and Rosetta Commons attempted to move novices from the periphery into core leadership roles.

Astropy has continued to evolve its leadership model, and in doing so has seeded decision making power, and task distribution to a more diverse set of participants. This has been achieved by adding new governance roles, and expanding the diversity of leadership positions that can be held: Astropy now has appointed roles such as core-package and package coordinators, release coordinators, and sub-package maintainers. Each of these roles are appointed at either a "lead" or a "deputy" level – with deputies expected to support and learn from more experienced leaders. A participant explained this granularity is both necessary for the size of the project, and for recognizing and encouraging highly engaged contributors to take on specialized tasks [PA-1]. Over time, it is expected that deputies can learn enough from leads that they will eventually cycle into the position.

Rosetta Commons, in somewhat contrast, has a leadership board but no formally acknowledged roles for contributors that recognize leadership. Each sub-package or library repository has an owner, but as one participant explained ownership has more to do with who made the initial commit to a version control system than to who actually was going to assume responsibility for the package long-term. Newcomers to Rosetta typically only learn the software through a bootcamp experience, and are then mentored within their own lab on best practices for maintenance. The lack of a formal governance structure beyond this board was seen as a dilemma that reflected the nature of Rosetta as an academic project whose sustainability depended not on students, but on an infrastructure team.

9 | DISCUSSION

We turn now to interpreting how the arrangement of social relations, shared artifacts, and institutional rules impact the sustainability of research software. Drawing on the work of James Carse (2011), we describe research software sustainability as being the product of games that have finite and infinite properties. The design of and funding for research software often resembles a finite game. To improve the sustainability of a research ecosystem that depends upon software we argue that these rule sets should, instead, reflect an infinite game.

9.1 | Finite and infinite games in software sustainability

Carse argues there are exactly two kinds of games: the finite and the infinite. Finite games are bound temporally and spatially; they have rigid rule structures which govern the state of play; and, most importantly, they have definitive winners and losers. Players of a finite game are keenly aware of the rules, and often make strategic decisions and manipulate rules in order to win (or lose). Most contests, sports, board games, and video games are designed as finite games.

Oppositely, infinite games have rules that exist only for the sake of continuing play. Players of an infinite game are incentivized to adjust their actions, reconfigure boundaries, share power, and negotiate the terms of engagement for the sole purpose of keeping the game alive. The roots of an infinite game are in cooperation and furthering play for the sake of a collective good. Energy grids, ecosystem services, and politics are often (though not always successfully) identified as examples of an infinite game. In each setting, actors are incentivized to adjust rules, create policies, and further play through cooperative actions.

The ludic properties of research software development are similar to those Carse outlines, but we argue

that research funding bodies unwittingly incentivize finite rather than infinite players. For example, the success of many scientific projects are predicated on playing a finite game - A researcher proposes a grant, executes research that successfully delivers on the promises of that grant, and then uses results from the grant to further obtain new funding, publications, and prestige which can further their own research. Developing sustainable software is at odds with the finite player's motivatations in a scientific enterprise. Investing time, energy, and precious grant funding in the maintenance of a software contribution that can be used by a broad community is at odds with winning new grant funding. Much of the contemporary landscape of scientific software development resembles a finite game rather than an infinite game.

Software projects like Rosetta and Astropy resemble an infinite game. Contributors to these projects have rejected winning and have instead invested time, energy, and organization activities towards 'continuing the state of play' - that is, they have taken deliberate collective action to keep a software package alive and useful to a broad community of researchers. This is despite the fact that career advancement, winning grants to maintain cooperative work, and receiving credit for their contributions are rare. As we noted at the beginning of this paper, this is despite the fact that there is increasing evidence for the immense value and productivity of sustainable software to fuel new discoveries, and enable efficient response to critical scientific phenomena.

What then should funding agencies do to invert the state of games played by scientists developing and using research software? We believe this requires, at minimum, the following:

- Credit and acknowledgement for activities related to curating software, developing, maintaining, and supporting software and hardware, are necessary for sustainability. Funding agencies should not just recommend, but require that grantees acknowledge what open-source software is used to produce a novel finding, conduct an experiment, or produce new knowledge.
- Research institutions, including academic and national laboratories, also need to give equal weight in tenure, promotion, and hiring practices to all of the activities necessary to cooperatively produce scientific software. This requires a radical change in not just the way that funding agencies reward these activities, but fundamentally shifting expectations for reliably educating and sustaining a research and development workforce.
- Data management plans were an important first step to developing a more sustainable research ecosystem, but to ensure reliable management for the long-term access to important software contributions, research funding agencies need similar software sustainability plans. These policy instruments should require that grant awardees document and make clear why new software needs to be developed, how it will be shared openly, and ways that this software will remain accessible into the future.

10 | CONCLUSION

In this paper, we have provided preliminary analysis of an ethnographic study of software sustainability across two long-standing projects - Astropy and Rosetta Commons. We demonstrate that efforts to sustain a software ecosystem within a competitive research environment requires the structuring of formal institutional rules that have ludic properties - those resembling finite games are common, and often successful in the short-term; those resembling infinite games, such as Astropy and Rosetta, are successful in sustaining software projects despite a rule set that increasingly disadvantages cooperative software development. In future work, we hope to significantly expand the description of how these ludic properties are applied across research software development activities, and further describe the unique institutional forms, social relations, and shared artifacts that are present in successful research software projects.

ENDNOTE

[1] See Sean Furguson's tweet for an extended commentary

REFERENCES

Bietz, M. J., Paine, D., & Lee, C. P. (2013, February). The work of developing cyberinfrastructure middleware projects. In Proceedings of the 2013 conference on Computer supported cooperative work (pp. 1527-1538).

Broman, K., Cetinkaya-Rundel, M., Nussbaum, A., Paciorek, C., Peng, R., Turek, D., & Wickham, H. (2017). Recommendations to Funding Agencies for Supporting Reproducible Research. In *American Statistical Association* (Vol. 2). Retrieved from https://www.amstat.org/asa/files/pdfs/pol-reproducibleresearchrecommendations.pdf.

Bollen, K., JT. Cacioppo, RM. Kaplan, JA. Krosnick, JL. Olds, and H Dean. 2015.*Social, behavioral, and economic sciences perspectives on robust and reliable science.* Technical Report. National Science Foundation.

Boscoe, B. M. (2019). *From blurry space to a sharper sky: Keeping twenty-three years of astronomical data alive.*

Campbell, M. L., DeVault, M. L., Diamond, T., Eastwood, L., Griffith, A., McCoy, L., ... Weatherbee, D. (2006). *Institutional ethnography as practice*. Rowman & Littlefield Publishers.

Carver, J. C., Gesing, S., Katz, D. S., Ram, K., & Weber, N. (2018). Conceptualization of a US Research Software Sustainability Institute (URSSI). *Computing in Science & Engineering*, *20* (3), 4–9.

Carse, J. (2011). *Finite and infinite games*. Simon & Schuster.

Das, R., & Baker, D. (2008). Macromolecular modeling with Rosetta. *Annual Review of Biochemistry*, *77*, 363–382.

Halfaker A, A Kittur, J Riedl. (2011, October 3–5) Don't bite the newbies: how reverts affect the quantity and quality of Wikipedia work. In Proceedings of the 7th International Symposium on Wikis and Open Collaboration. Mountain View, CA.

Hars, S. O. (2002). Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce*, *6*(3), 25–39.

Howison, J. 2015. "Sustaining Scientific Infrastructures: Transitioning from Grants to Peer Production (Work-in-Progress)." iConference 2015 Proceedings. Retrieved from http://hdl.handle.net/2142/73439.

Howison, J., & Bullard, J. (2016). Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature. *Journal of the Association for Information Science and Technology*, *67*, 2137–2155.

Hwang, L., Fish, A., Soito, L., Smith, M., & Kellogg, L. H. (2017). Software and the scientist: Coding and citation practices in geodynamics. *Earth and Space Science*, *4*(11), 670–680.

Moher, D., Naudet, F., Cristea, I. A., Miedema, F., Ioannidis, J. P. A., & Goodman, S. N. (2018). Assessing Scientists for Hiring, Promotion, and Tenure. *PLoS Biology*, *16*(3), e2004089.

Park, H., & Wolfram, D. (2019). Research software citation in the data citation index: Current practices and implications for research software sharing and reuse. *Journal of Informetrics*, *13*(2), 574–582.

Khatib, F., Cooper, S., Tyka, M. D., Xu, K., Makedon, I., Popović, Z., & Baker, D. (2011). Algorithm discovery by protein folding game players. *Proceedings of the National Academy of Sciences*, *108*(47), 18949–18953.

Muna, D., Alexander, M., Allen, A., Ashley, R., Asmus, D., Azzollini, R., ... & Bilicki, M. (2016). The Astropy Problem. arXiv preprint arXiv:1610.03159.

Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., Conseil, S., ... VanderPlas, J. T. (2018). The Astropy project: building an open-science project and status of the v2. 0 core package. *The Astronomical Journal*, *156*(3), 123.

Rankin, J. (2017). Conducting analysis in institutional ethnography: Guidance and cautions. *International Journal of Qualitative Methods*, *16*(1).

Ram, K., Boettiger, C., Chamberlain, S., Ross, N., Salmon, M., & Butland, S. (2018). A community of practice around peer review for long-term research software sustainability. *Computing in Science & Engineering*, *21*(2), 59–65.

Renfrew, P. D., Campbell, G., Strauss, C. E., & Bonneau, R. (2011). The 2010 Rosetta developers meeting: Macromolecular prediction and design meets reproducible publishing. *PloS One*, *6*(8), e22431.

Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., ... Conley, A. (2013). Astropy: A community Python package for astronomy. *Astronomy & Astrophysics*, *558*, A33.

Rohl, C. A., Strauss, C. E., Misura, K. M., & Baker, D. (2004). Protein structure prediction using Rosetta. In J. Abelson, M. Simon, G. Verdine, & A. Pyle (Eds.), *Methods in enzymology* (Vol. 383, pp. 66–93). Academic Press.

Shaikh, M., & Henfridsson, O. (2017). Governing open source software through coordination processes. *Information and Organization*, *27*(2), 116–135.

Siepel, A. (2019). Challenges in funding and developing genomic software: roots and remedies. *Genome Biology*, *20*(1), 147.

TeBlunthuis, N., Shaw, A., & Hill, B. M. (2018, April). Revisiting the rise and decline in a population of peer production projects. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (p. 355). ACM.

Thornton, P. H., & Ocasio, W. (2008). Institutional logics. In R. Greenwood, C. Oliver, R. Suddaby, & K. Sahlin-Andersson (Eds.), *The Sage handbook of organizational institutionalism* (pp. 99–128).

Vaismoradi, M., Jones, J., Turunen, H., & Snelgrove, S. (2016). Theme development in qualitative content analysis and thematic analysis. *Journal of Nursing Education and Practice*, *6*(5), 6–7.

Wheeler, D. A. (2004). SLOCCount–a set of tools for counting physical Source Lines of Code (SLOC). Retrieved from: http://www.dwheeler. com/sloccount.

How to cite this article: Weber N. Finite and infinite games: An ethnography of institutional logics in research software sustainability. *Proc Assoc Inf Sci Technol*. 2020;57:e281. https://doi.org/10.1002/pra2.281