Tales of Transitions: Seeking Scientific Software Sustainability (Pre-print)

JOHANNA COHOON, CAIFAN DU, and JAMES HOWISON, University of Texas at Austin, USA

Software is crucial to science, but sustaining projects for long term impact is challenging. Scientists and funders recognize the affinities between academia and peer production ("the open source way" underlying the success of open source software development) and see peer production as a promising route to sustainability. We studied 120 scientific software projects funded by research grants and which were encouraged by their funder to develop a sustainable community around their open source code. Using interviews and content analysis of online presences, we studied the projects over a seven year period, from receiving grants around 2014 through the typically three years of funding, and for up to four years following the funding. We make three contributions: First, through our taxonomies of organizational forms and of organizational change, we paint a landscape of the variety of forms that scientific software development can take on and transition between (including peer production), providing language for discussion in the literature and among communities of practice. Second, we use these taxonomies to describe multiple routes to sustainable software development. Finally, we discuss the challenges and strategies involved in sustaining scientific software, including choices not to pursue peer production at all.

CCS Concepts: • **Software and its engineering → Open source model**; • **Human-centered computing → Empirical studies in collaborative and social computing**.

Additional Key Words and Phrases: open source software development, scientific software, qualitative methods, collaboration, incentives, infrastructure

1 INTRODUCTION

Software valuable to science is often difficult to develop, requiring deep understanding of scientific theory, workflows, algorithms, and cutting edge performance strategies, but successful software can advance science, learning, and collaboration [1, 28, 36, 43, 55]. Accordingly, science funders (and communities of researchers through peer review) support software work in science with grants. Yet grants are relatively short term funding commitments and there is concern that grant-funded software languishes once the grant ends. Science policy thus seeks approaches to encourage the sustainability of grant-funded research software[20, 54, 55]. While commercialization is an option [14, 74], the success of open source software development—and its affinities with the ethos of science—is inspirational [23, 54, 55].

While open code itself is important, the novel organizational form associated with open source production is especially desired. Whether referred to "the open source way," "open collaboration," or "peer production" the image is of communities of users collaborating to build, sustain, and enhance the tools they use in their work [2, 5, 6, 37, 51, 79]. These collaborators work with high autonomy, openness to new contributors, commitment to shared learning, and shared responsibility for success. The community often undertakes this effort without a centralized budget or investment.

Early work on open source emphasized volunteerism [e.g., 63], implying that contributors are not paid while contributing work. Decades of subsequent work has corrected this view, making clear that the majority are contributing with time paid for by employers who benefit from the software [29, 33, 80]. Other motivations for developers to contribute include still further extrinsic motivations, including reputational enhancements. Other early visions of open source have gained nuance as well. For instance, while some images of open source collaboration—including the label "peer production" itself—emphasize a lack of hierarchy, projects can be quite hierarchical (including

Authors' address: Johanna Cohoon; Caifan Du; James Howison, jhowison@ischool.utexas.edu, University of Texas at Austin, USA.

those run by a "BDFL", a Dictator For Life whose Benevolence is, in fact, often loudly disputed) [e.g., 25, 66, 67].

Rather than non-hierarchical projects driven by unpaid volunteers recent conceptualizations of open source work emphasize the importance of autonomy for participants and for the code itself. Participants largely decide for themselves what work they will undertake, offering patches that may not be accepted by projects, demonstrating *a work first, choose later* approach described as "shared leadership" [30, 66]. Autonomy for the code is guaranteed by open licenses, denying even dictators the ability to restrict use, or access to the source code to anyone. An open licence also ensures the ability to "fork a project," taking a full copy of the codebase to develop independently or as a base to attract like-minded contributors to form a new community.

Collaboration technologies underlie, but do not guarantee, the organizational form of peer production. From email discussion lists, source code diffs and patches, source code repositories, issue management and pull requests, to continuous integration and bots for work [41], open source has embraced and in many cases produced the collaboration technologies underlying our world. The low cost and persistence of these work environments means that work can precede organizing and organizations, reducing the need for money to be invested and the protection of interests that results in formal contracts, advance agreements, and fixed contracts money implies [82]. Work and artifacts can seek forms of organization able to sustain them over time.

Much of peer production is familiar to discussions of science. In fact, research has established that "the open source way" was directly inspired by the organization and experience of academic work [8, 60, 75]. Certainly the expressed aspirational values of science are well aligned with open source and peer production [18].

Yet research also shows that scientific research has its own well-established and institutionalized organizational forms, many of which look quite different from peer production [48, 49, 57, 68, 71, 78]. Collaboration technologies are also diverse in science, ranging from low use through to advanced, innovative use, such as in the development of Collaboratories [31, 52, 56] and distributed collaborations with world leading technology and scale, such as the Large Hadron Collider. Accordingly, sustainability through peer production likely requires considerable change, certainly in organization and likely in collaboration technologies. Thus, on top of the challenges of realizing sustainable peer production at all are layered challenges of changing practices and changing, or learning to coexist, with existing institutions.

Accordingly our research questions are these:

(1) How is software work funded by grants organized? How does that organization change during and after the period of a grant?
(2) What routes to peer production did our projects pursue?
(3) What challenges exist, and how are they addressed, in seeking sustainability through peer production in Science?

2 RELATED WORK

Previous research has addressed closely related topics, both inside science and outside. Our research drew on this literature throughout our project and we attempt to highlight similarities and contrasts in our results and discussions of findings.

Research on open source software development has sought to describe overall models of organization [5, 37, 51] and governance[66, 67]. Studies have identified why participants are motivated to participate [62] as well as the inputs, processes, and outcomes of open source projects [22]. Recent work in CSCW has highlighted the labor of scaling and maintaining large projects, observing the significant community work involved [35]. Ironically, given the vision of sustainability investigated

in this paper, while early research largely sought to explain and learn from the success of open source [21, 64, 72], more recent research has highlighted challenges to sustainability, including an under-provision of labor overall and high burnout rates, particularly among those doing long term maintenance work [29]. There has likely been a fall in the overall number of active open source projects [26].

Studies of work in science have examined software within a larger category of infrastructures, including communication platforms, scholarly publishing, data repositories, and software. In particular, this research identified tensions between work seen as research versus infrastructure [13], and the limits of systems of academic credit to reward these efforts [11, 61] joining strong findings on the relative invisibility of infrastructural work [50, 70].

These findings were strengthened by specific studies of software work in science, highlighting software sharing among scientists[40], troubles with reputation motivation [38, 39], and identifying specific forms of organizing prevalent in scientific software work [38, 65]. Bietz and colleagues [9, 10] specifically examine ongoing maintenance of scientific software specifically; their work described the similarity between a period of purported maintenance and the work of initial creation. Those working on the software "synergize" across projects and scientific fields, seeking to enroll additional sources of support while maintaining software in the face of a shifting software ecosystem. Darch and Sands [24] studied challenges to the open source approach to managing uncertainty in the data management needs of the LSST research project. Trainer et al. [77] studied the "extra work" required to transition from personal infrastructure to a community resource. These results provide scientific context to long-term findings about the difficulty of sustaining software projects in general [4, 58] and the surprisingly high costs of maintenance phases in software lifecycles [3, 12].

Finally, practice-oriented literature discusses how to establish productive open source software work, both in general and specifically in science. For example, Fogel [34] approaches open source software production systematically, attending to its technical, social, organizational, and even political aspects. A series of Workshops on Sustainable Software for Science: Practice and Experiences (WSSSPE) represents the proliferating initiatives in science that seek to address the collective challenge of sustaining reusable software for scientific research [e.g., 44]. These workshops have also distilled expectations for software project infrastructure, including public repositories, mailing lists, continuous integration, and wikis. Similar to general guidance on running open source software projects, scientific software practitioners also seek to address the social and economic challenges in sustaining open software development, and they additionally attend to specific requirements raised by scientific research [7, 27, 45, 46, 59]. We drew on these guides as we built our content analysis scheme, described below.

3 METHODS

To understand how projects are organized and establish sustainability possibly by transitioning to peer production, we conducted a qualitative study of grant-funded scientific software work. In this section, we describe our methods: content analysis of project web presences and semi-structured interviews. Initially, we focused our efforts on understanding transitions to peer production. As it became evident that projects also tried alternative approaches to sustainability, we reoriented to include these possibilities in our analysis.

3.1 Sample

Our sample was drawn from NSF's program "Software Infrastructure for Sustained Innovation" (abbreviated as SI2). The SI2 funding program intended to create "sustainable software communities" and required each project to have a "Sustainability Plan" [54]. Thus, we interpreted SI2 grant

recipients as a useful population through which we can study the sociotechnical changes that occur during a transition for sustainability.

The results presented here are based on a study of 120 projects funded through 81 different grants, selected from a pool of over 250 grants funded by the SI2 program. This began as a theoretical sample of 15 grants chosen to represent what we expected to be a variety of research topics, team sizes, and goals (e.g. to contribute to existing software or develop a new package). To ensure that projects had sufficient time to achieve transitions and to ensure we had sufficient empirical cover to avoid missing sparse success, we expanded our initial sample to include all grants that began before 2014. Grants funded multiple software projects; we identified the 120 projects through the content analysis process described below.

3.2 Content Analysis

We reviewed current and archived websites associated with the funded software projects, recording information that the literature suggested might impact the success of a peer production community (e.g., the presence of a public code repository). We established a coding scheme and initially attempted to train a group of about 10 research assistants on its use. Through this attempt, we clarified our coding scheme, but training costs were high and research assistants made few unsupervised contributions before they left the lab. Our interests turned toward establishing in-depth histories for the software projects and, as the work became more qualitative and narrative, a large team approach became infeasible. The present authors then took over data collection and analysis fully.

For each grant, the authors collected publicly available documents and web pages. Beginning with the published grant abstract located on the NSF website, we then discovered other relevant web pages by conducting Google searches with keywords like grant numbers, titles, names of software, and names of personnel mentioned on the grants (including any linked collaborative grants). We found project homepages, SI2 PI meeting posters, published papers, PI institutional websites, mailing list archives, and source code repositories (e.g. GitHub, Sourceforge, Bitbucket). As we studied the projects' web presences, we adopted a persona of a potential contributor.

As we read and discovered documents, we wrote memos about the project, noting who was involved in the work. We used the documents we collected as sources of insight into the way that the project was organized. We applied our coding scheme, recording data in RDF about individual web pages as well as the project as a whole. Through discussions about the data and our research process, we continued to refine our coding scheme. Ultimately, this analysis yielded language to describe different organizational forms involved in grant-funded scientific software development. This language, detailed in Section 4.1, articulates a range of possible organizational configurations, reported below, and the transition methods we observed.

The RDF data was queried using Data.World (https://data.world/) and figures representing transitions and website activity were developed using the R package ggalluvial [16, 17].

Identifying active websites. As we refined our coding scheme, one addition we made was to log whether or not a discovered website appeared to be active, in the sense that the work it was describing was ongoing. Relying on commit histories, other dated content, and archives on the Internet Archive (archive.org), when possible, we took note of the date the website appeared to become active or inactive. Taking into account all relevant URLs for a project, we then assessed whether or not a project had *any* active websites at the start of the grant period, the end of the grant period, or after the grant period. These data are represented in Figure 4.

3.3 Interviews and Thematic Analysis

To enrich our analysis, we interviewed project leaders and contributors from a subsample of our projects. We sought detailed explanations of work episodes (e.g., when a person outside the core team contributed code) and descriptions of how the project organization has changed over time. We conducted an initial set of nine interviews about five projects in 2018, including two projects that were not SI2 funded but that were known to the authors as having undergone or attempted an organizational transition. A second round of 39 interviews with SI2-funded participants began in 2020 and concluded in 2021. This second round occurred after we developed our taxonomies. In these later 39 interviews, we spoke with representatives of 24 projects that were funded by 22 grants. Having already assigned 1) organizational configurations at the start and end of the grant period and 2) transition methods to the software projects in our sample, we were able to deliberately schedule interviews together in "pods" that highlighted similarities or differences across the configurations. Each pod included at least one project that we assessed as having been organized as peer production at some point, together with projects we thought had not.

We recorded notes during and after each interview, discussing highlights and questions with one another. The authors listened to the interviews again, recording notable quotes in an index that was then discussed to identify prominent themes and analytical questions. One author reviewed the full index for thematic analysis [15]. Attending to the conditions for peer production, challenges for sustainability, and resources for sustainability, the author categorized the quotes. These categories included *keeping up with technology*, *healthy, connected ecosystem*, *competition and control*, *collaborators/sources of ideas*, *integrating contributions*, *maintenance work*, *best practices*, *developing for contributions*, *developing for use*, *establish shared expectations*, *upskilling/mentoring contributors*, *contributor turnover or retention*, *money as a critical resource*, *parent organizations*, *charging money*, *balancing science and engineering*, *promotional/community work*, *governance*, *RSEs*, *transitions*, and *relationships with industry*. Quotes from projects that successfully transitioned during their grant period were reviewed once more for their relevance to any category. Analytically iterating through the data once more, one author considered the relationships between categories, combining and refining them to establish a set of themes. These themes represent the strategies and concerns of software projects in our sample. The other authors critiqued and added to the themes. All authors sought to find counter examples to the emergent findings, challenging our conclusions. The results of this process are presented in Section 5.

We "disciplined our imagination" and interpretations [81] by discussing and presenting results as they developed with the communities of our participants. In particular, we presented results at workshops, prepared white papers, and gave keynotes and participating in discussions at practitioner events. At the conclusion of our interviews, we often answered participants' questions, summarizing results. This engendered reflection, additional stories, pointers to resources, and sometimes polite critique. Nonetheless, the conclusions in the paper are those of the authors, and not our participants.

3.4 Theoretical Framework

Throughout our analysis, we drew on the peer production literature as well as the theoretical concepts of organizational configurations and genre.

Using the peer production literature, we established expectations for the changes in infrastructure and work practices that we would see as projects transitioned to peer production. In particular, the outcomes of the First Workshop on Sustainable Software for Science: Practice and Experiences identified a number of technological "needs" in peer production: "public source-code hosting, mailing lists, documentation, wikis, bug trackers, software downloads, continuous integration,

software quality dashboards, and of course, a general web presence to tie a project's channels and artifacts together," [44][page 6].

Exploring the organizational science literature, we were sensitized to the concept of organizational configurations [32, 53]. Configurations are potential structures that an organization might adopt [53], identified by finding resemblances among various organizations [69]. In finding similarities among organizations, a configuration may be identified and predictions about that configuration's success in certain circumstances may be made [32, 47, 69].

As we sought similarities among projects to recognize organizational configurations, we also drew on the concept of genre. Genres are recognizable by their form and function, and are developed over time through a community's communication practices [73, 83]. They are therefore useful lenses through which to understand changes in software projects' collaboration and infrastructure. We considered the "communicative purpose" [73][page 58] of each grant's web pages and attended to their visual style. Thus, website features like a university seal were interpreted not just as imagery associated with certain organizational configurations, but as indications of the project's priorities and their work practices.

4 FINDINGS

Through content analysis, thematic analysis, and regular "dialoguing with the data" [19, page 106], we developed two taxonomies, applied these to describe projects' changes over time. After introducing these taxonomies and applying them to our projects we report the results of our thematic analysis: challenges and strategies of sustaining scientific software, differentiated between concerns for transitioning to peer production and generally maintaining for sustainability.

4.1 Taxonomies of Transition

4.1.1 Organizational Configurations.

How software projects were organized. By identifying patterns in the codes describing the projects and attending to presented genres, we recognized six organizational configurations within which software was being produced. We then assigned these configurations to the software projects in our sample. We labeled 45 projects as ending the grant period in a lab, 35 as ending in a tool group, 24 as ending in a peer production community, eight in an author group, four in a business, and four in a consortium, as shown in Figure 1. We describe these below.

Here, we define these configurations. While we developed many codes during our analysis, in this paper we focus on those we used to distinguish among the configurations. Additional data from our content analysis are shared to demonstrate genre features.

Peer production communities. Our sample of 120 SI2 supported projects included 24 peer production communities at the end of their grant period. We identified them by their possession of five key characteristics:

(1) Self-identification as a community: Peer production communities were the easiest to identify as they were often self-proclaimed community driven projects.
(2) An established group website: We saw that peer production communities used their websites to share information with current and potential contributors. These websites listed current and past contributors and they all included a link to a public repository. Their web presence also often included a user or developer forum (16 of 24 projects) and half of the peer production websites indicated that a user workshop had been held, indicating a community oriented genre.

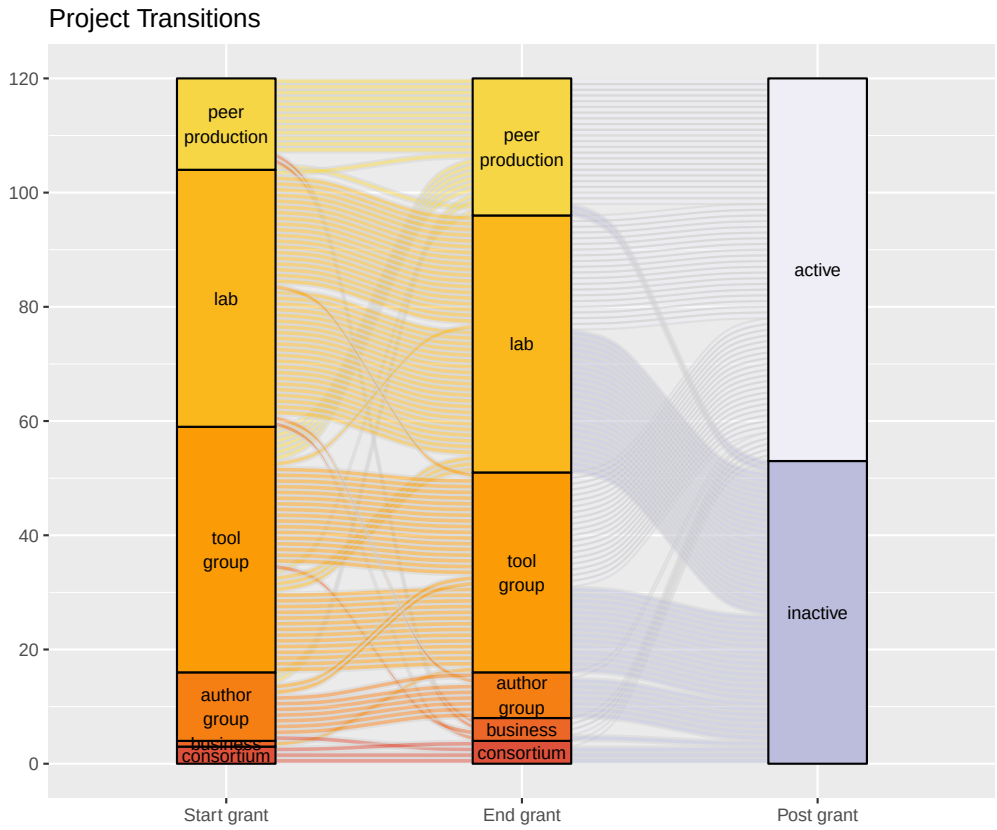

Fig. 1. Configurations at the start and the end of grant periods. Graph using ggalluvial [16, 17]

(3) A formal name for the group: Peer production communities in our sample referred to their collaborations by the name of the software they were producing.
(4) A clear invitation to contribute: Nearly all of the peer production communities in our sample extended obvious invitations for potential participants to contribute (21 of 24 projects). Two of the three that did not explicitly invite invitation still provided descriptions of how to make a contribution. The third specified that it is maintained by volunteers and our knowledge of the grant work made it clear that this particular project was peer production.
(5) Singular interests: The peer production communities we saw were focused on a singular software project, rather than multiple research or software projects.

Labs. Our sample included 45 projects where the software was maintained by a lab. We defined labs as possessing four key characteristics:

(1) An established group website: Labs usually had a website listing their ongoing activities and personnel. Frequently, this website listed both past and present members (usually graduate or undergraduate students). Publication lists and university affiliations were also commonly seen on lab websites. Across URLs, labs' web presences most often included a public repository (33 of 45 projects) but rarely included community oriented content like a forum (six of 45 projects) or evidence of a user workshop (seven of 45).

(2) A formal name for the group: The labs we identified were usually named for their founder or for their research topic.
(3) No invitation to contribute: Only eight of the 45 SI2 funded projects that we labeled as a lab invited outside contributions and only six included instructions on how to make those contributions. In most situations, by adopting the persona of a potential contributor we inferred that to contribute to lab activities we would need to apply to a degree program or contact the PI for employment.
(4) Diverse research interests: We saw that labs often have an array of research interests. The SI2 funded work was usually not the lab's only ongoing project.

Tool groups. Our sample included 35 projects we identified as what we called "tool groups" by the end of their grant period. In our taxonomy, tool groups possess four key characteristics:

(1) An established group website: Grants we identified as tool groups usually had a website that shared information about their group and software. These websites usually listed publications related to their work and linked to repositories or downloads of their software. We also often saw a list of members posted to the tool group's website. 26 of 35 tool groups had a public repository and 11 maintained a user or developer forum. An equal number of projects showed evidence online of having held a user workshop.
(2) A formal name for the group: We saw that rather than naming themselves for the group's founder or research area, tool groups named themselves for their software. While members include representatives from different universities and organizations, these identities typically fell to the background as the group promoted their collective software project.
(3) Limited outside contributions: While some tool groups did invite contributions (nine of 35 tool groups), fewer had specific instructions for doing so (seven of 35 tool groups). To join a tool group, an interested developer would usually need to apply or obtain permission of some sort. Tool groups often had paid staff dedicated to certain tasks.
(4) Singular software development interests: Tool group were usually dedicated to developing a single piece of software.

Author groups. We categorized eight projects from our sample as author groups at the end of their grant period. Author groups possess three key features:

(1) A minimal web presence: Author groups had very little web presence—three of the eight in our sample had no project homepage at all. Only three author groups had a public repository and only one had a forum for users or developers. None showed any evidence of having had a user workshop.
(2) No formal name for the group: Participants published or presented together without naming or otherwise formalizing their collaboration.
(3) No invitation to contribute: Author groups did not include information on posters or in papers on how readers could contribute to the project or become a member of the collaboration.

Business. We categorized four of the software projects in our sample as businesses at the end of their grant period. Businesses were easily identifiable by two characteristics:

(1) Revenue driven: Businesses sold their software/services, either for profit or as a non-profit business.
(2) Legal status: Businesses sometimes included legal information or information on tax-deductible donations on their websites, indicating that they have a formal legal status.

Consortia. We categorized four software projects as being consortia at the end of their grant period. Consortia were identified by two characteristics:

(1) A network of organizations: Consortia collectively took responsibility for work and goals, but operated as collaborations among multiple independent organizations.
(2) Identification as consortium: Consortia often identified themselves as such, using the term in their organizations' name or describing themselves as a collaboration among organizations.
(3) An established group website: Consortia had websites that described the members of their organization and their work. Consortia websites emphasized the role of collaboration among their members.

Consortia were particularly difficult for us to recognize without their self-identification. Often, consortia seemed to resemble tool groups because of their limited outside contributions and non-collocated members.

4.1.2 Transition Methods. Initially as we studied the changes software teams underwent, we were alert for efforts to change into a peer production community by "open sourcing" the code to initiate a project, introducing new workflows and tools, and building a community around the software. However, as we worked with our data it became clear that to properly consider routes to peer production, we had to ask which organization was taking responsibility for the software, in what configuration, and at what time. Analytically, this meant separating the organization and the organizational configuration, and considering that each may remain the same or differ around the software. In this way, we identified three modes of transition, shown in Figure 2: reorganization, hand-off, and migration.

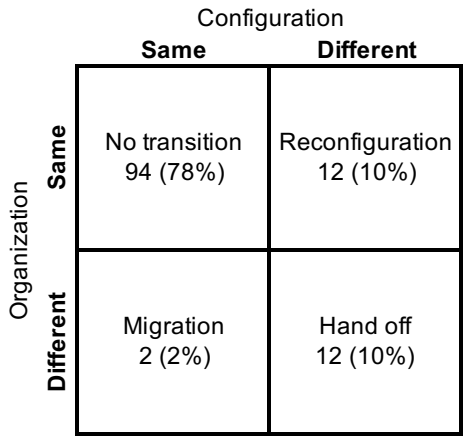

Fig. 2. Frequencies and Modes of Transition by Software Projects

Like the organizational configurations, we assigned a mode of transition or "no transition" to each software project in our sample. Most projects (94, 78%) underwent no transition during their grant period. Of the 120 software projects we examined, we coded 26 as undergoing any kind of transition. Only two projects underwent a migration. Twelve projects went through hand-offs and twelve also went through reorganizations. Below, we define these processes and provide narratives from our data to demonstrate a typical transition of each type.

Reorganization (same organization, different configuration). Reorganization occurs when an organization that develops a piece of software changes its configuration. In this case, the participants do

not change but the key characteristics of their configuration do. This means that the software never changes organizational hands, but the configuration surrounding it transitions by changing social structures and online infrastructure. Of the 120 software projects, 12 (10%) reorganized themselves by the end of their grant period (but not necessarily into a peer production community).

To illustrate, one project in our sample presented as a lab at the start of the grant period. This was demonstrated in part by the references to the project team as members of a specific university lab and by the fact that a student in that university lab owned the Github repository. Somewhat contrary to the lab genre, the project had a weak invitation for outsiders to contribute at the start of the grant, inviting readers to "send an email to the project faculty." However, during the grant period this invitation was strengthened greatly. The code moved to a Github repository owned not by a student or a lab account, but by an account named for the project. A new website was created; this new website included a page titled "Community" that had new information about how to integrate code and a welcoming message that shared how they are "excited" to grow their development team. Notably, the core contributors remained the same throughout this period. This strengthening of the invitation to contribute, the identification as a software group rather than university group, and their new web presence indicated a reorganization to peer production.

Hand-off (different organization, different configuration). A hand-off occurs when one organization with a given configuration develops software and then turns that software over to another organization with a different configuration. This means the people who undertook the software work at the start of the grant are different from those who assume maintenance responsibility at the end of the grant. In our sample, we saw 12 (10% of our sample) software projects handed off by the end of their grant periods to a new organization and configuration.

A hand-off in our sample transferred maintenance responsibility from an author group to a peer production community. This author group was composed of a PI and their student—this collaboration presented less as a lab and more as an author group because of their lack of web presence. The student was responsible for doing the coding work. They created a large PR in the receiving peer production community; this PR was met with friendly apprehension. Multiple core developers coached the student through revisions to their PR so that the code would better meet project standards. The student and developers communicated on Github but interviews indicated that some conversations happened over the phone and at meetings. The student attempted to avoid additional documentation work, citing their need to graduate soon, but developers insisted and the PI arbitrated the conversation. The hand-off concluded with the merged code, but developers indicated that they viewed the contributed feature as leading to more work because it would prompt users' desire for a followup feature.

Migration (different organization, same configuration). Migration occurs when responsibility for a piece of software is handed over from one organization with a given configuration to another organization with the same configuration. We saw only two examples (2%) of migration occurring during the funding period of any of our studied grants.

A software project that underwent migration was originally developed in a University Lab (Lab 1). A masters student worked to develop the code, eventually publishing on it. The publication contains a link to the Lab 1's website, demonstrating its status as the maintainer of the software. The PI at the first Lab then teamed up on an NSF SI2 grant with a collaborating PI, Professor X, from another university who runs their own lab. Professor X's student continued the software's development and earned their masters thesis based on their contributions. Professor X's lab and personal websites make reference to the software project. Open repositories discoverable online belong to Professor X's lab while University Lab 1's website makes little mention of the project. Continued maintenance of the code by Professor X's group is visible during the grant period through commits and there is

no evidence of further contribution from University Lab 1, thus a transition has been away from University Lab 1 and toward Professor X's lab. The project configurations are understood to both be labs because each lab has named itself after their area of expertise, they do not invite contributions, their websites indicate multiple interests. The students earning their degrees and publishing also made clear that both University Lab 1 and Professor X's group were labs.

No transition (same organization, same configuration). Software makes no transition when the organization that produces it neither changes its configuration nor hands over responsibility to another party. This means that participants responsible for software work at the start of the grant are the same participants responsible for work, if any, after the grant concludes. These participants demonstrate the same configuration throughout the grant period as well. This was by far the most common option seen in our sample. 94 (78%) of the projects in our sample made no transition. This includes projects that continued to have visible activity on their websites and projects that did not have an visible activity.

4.2 Transitions Observed

Figure 3 show all transitions observed in our dataset. Of the 26 software projects that underwent a transition, ten ended the grant period as a peer production community. These transitions were mostly reorganizations from tool groups to peer production (five projects). We also coded two tool groups as handing off their code to a peer production community, two author groups as doing the same, and one lab as reorganizing their project for peer production. We observed no migrations of code between peer production communities.

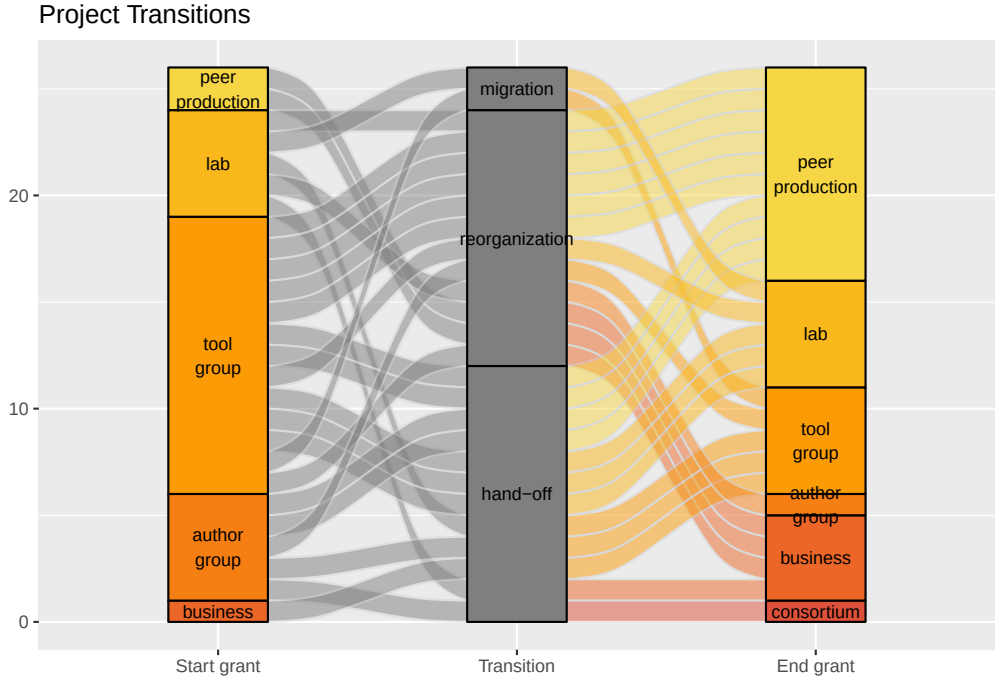

Fig. 3. Projects undergoing transitions. Graph using ggalluvial [16, 17]

While peer production was the most *transitioned to* configuration, the majority of the transitions we saw (16/26 projects that transitioned) were toward some other configuration. Five software projects transitioned to a tool group, five to a lab, four to a business, one to an author group and one to a consortium. We observed 12 software projects engage in a hand-off of their code and another 12 in reorganization. Migration was rare—we only coded two projects as migrating their code to another organization with the same configuration.

4.3 Ongoing activity

Ongoing public activity 2-3 years after the grant period was visible in over half the projects, including almost all the peer production projects, almost all business projects and approximately half the lab and tool group projects. The tool group projects were the least likely to have publicly visible ongoing activity, although these are the most likely to have ongoing activity that is not publicly visible (as they do not typically have project specific websites, with visibility coming primarily through publications and code releases). Given that some projects may be active but without updating websites, we interpret this as a floor for ongoing activity.

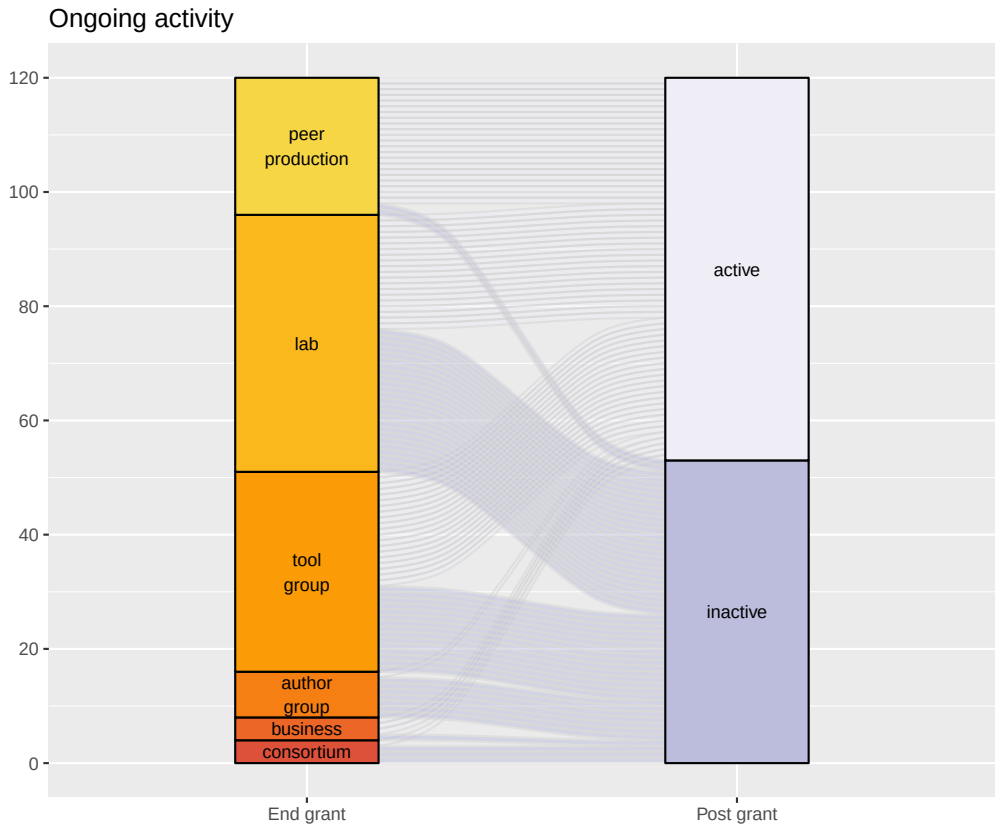

Fig. 4. Configurations and whether projects displayed publicly visible ongoing activity in the 3 year period following the end of the grant. Graph using ggalluvial [16, 17]

Overall, we think this rate of ongoing activity is fairly high. While direct comparisons are difficult, this rate is roughly comparable to the five-year failure rate of VC backed startups quoted in news media, substantially higher than the success rate of open source projects in general, thought to be as low as under 10% [22, 64] and high when considered by the NSF's mission to fund "high risk-high payoff" research. Nonetheless we interpret these results cautiously, as our assessment of ongoing activity was binary and we did not attempt to decide if the projects were truly sustainable in the sense of completing the work needed to keep software scientifically useful.

4.4 Routes to Peer Production

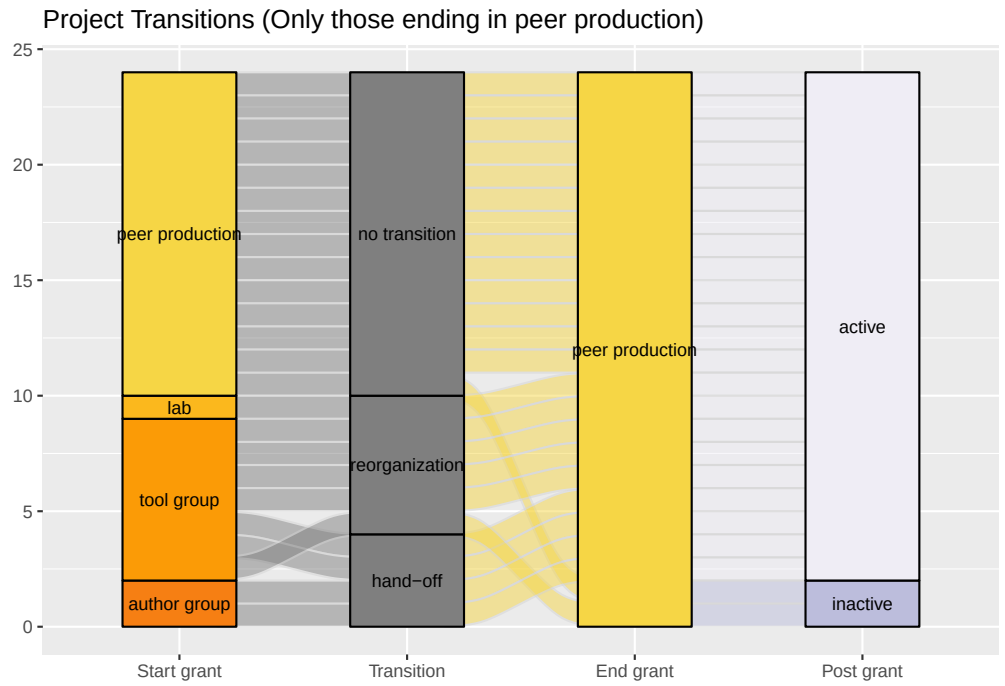

Fig. 5. Projects ending in peer production. Graph using ggalluvial [16, 17]

We found 24 projects that ended as peer production, and these projects had a relatively high level of sustained activity in the approximately three year post-grant period, only two projects appearing to be inactive.

The most frequent route to peer production was to begin as peer production, occurring in 14 of these projects. The second most frequent route to peer production was from a tool group (five by reorganization, and two through a hand-off to an existing peer production community). We saw two author groups handing-off artifacts to existing peer production communities, and a single lab that reorganized to peer production.

Of the two projects that appeared to be inactive after achieving peer production one begin the grant as a peer production project, and the second was a reorganization from a tool group.

5 CHALLENGES AND STRATEGIES FOR SUSTAINABILITY

Aligned with the literature on open source software development, our data showed that projects establish sustainability by creating value they leverage for resources. As the taxonomies above demonstrate, however, these projects are not all the same and their methods for creating and leveraging value consequently differ. Using these taxonomies and drawing on our interviews, here we discuss the strategies that scientific software projects use and the challenges that must be addressed as they maintain for sustainability. While we focus on sustaining peer production, we draw comparisons across configurations, especially attending to the contrast in approaches between labs and peer production. We also present evidence that shows transitions to peer production are

not an escape route from other maintenance responsibilities. Indeed, transitions incur additional costs that might not be recovered. The uncertainty of this return is a primary concern for sustaining or transitioning to peer production.

5.1 Maintaining for Sustainability

Regardless of configuration, scientific software projects must generate value and use that value to sustain their work. In this section, we describe this process in peer production communities and compare it to the approaches we saw other configurations employ. Maintenance challenges and strategies were influenced by the scientific context.

5.1.1 Sustaining Peer Production. Peer production projects primarily sustain their work by converting users into developers who contribute code that aligns with the organization's standards. This process is not always fruitful, however. Eghbal [29] describes *extractive* code contributions as those where the costs of reviewing and merging the contribution are higher than the benefit of the code to the project. Because benefit is reaped over time, it is difficult to identify extractive contributions in advance. During the investment, core developers must base their actions only on a predicted outcome. Consequently, sustainable peer production relies on core developers' ability to recognize extractive contributions in advance.

Developers we interviewed expressed concerns about these kinds of outside contributions, but sometimes they overcame their apprehension. One core developer of a peer production project told us about a contribution he integrated with reservation. "Part of the problem," he said, is that if a user tries it and it doesn't work, we are going to hear about it. Maybe [the contributor] hears about it too, but he is...not on the line. In this scenario, the developer integrated code that they perceived as prompting users' need for further features; they imagined the contributed code as being useful on a trial basis, but that when "it comes time to do the real thing...the program would say 'Oops, not implemented.'"

This developer described a case where their concerns about future costs were overruled by their belief that grant-funded contributions should be integrated. Because federal grant applications are peer reviewed, they perceived it as their responsibility to add a community-supplied feature. They, like others in our sample, also emphasized the need to welcome contributions so that the software would have a strong community around it. In spite of their concerns, they therefore invested significant time in integrating this contribution. Their onboarding practices as including a lot of "patience" and "purposeful praise" throughout iterative rounds of feedback. This work can be overwhelming at times for the core developer, unlike when their established collaborators make code changes: "I mean, if it's 100 lines of code from one of my co-authors, I know there's not all that much for me to do."

The onboarding and integration practices used to convert users into developers required core peer production developers to upskill their contributors, promote community, maintain a user base, and—to make these efforts possible—seek funding.

Upskill Contributors. Because of the specialized nature of the work, expert labor was difficult to find for the scientific software projects in our sample. Many of our interviewees described their efforts to upskill students and contributors, coaching them on best practices, communication style, and infrastructure use. Research Software Engineers and professional developers seemed to assume this mentorship role most often, educating contributors at all levels. One interviewee described how the parent organization to their software project employed two senior research software engineers who taught him best practices like documentation. He added, "They pay attention to that stuff. I would not know about pull requests and unit tests and all this were it not for them." PIs also frequently mentored their students through their code contributions. On many occasions, this was,

in part, an exchange for their cheap labor. One interviewee recalled that when they were a student they were told, "If you are willing to work on this, I will be happy to guide you."

Promote Community. Interviewees who sought to maintain peer production communities repeatedly mentioned that contributor turnover was a threat to their sustainability. As a strategy to mitigate this turnover and attract high quality contributions, software projects adopted community development practices to increase the project's visibility and create a hospitable culture. This work included evangelizing the project during conferences, workshops, and courses. Additionally, any interaction between a developer and a contributor was opportunity to encourage retention and promote community. Project owners described their strategies for enfolding newcomers into the group; this included raising expectations for code contributions slowly over time, using hospitable language, and adding features that meet new users' needs—regardless of whether or not that feature was planned for earlier. These strategies were also made evident through projects' web presences; for instance, after months of coaching a student through a PR, one developer raised their expectations for future revisions, requesting more extensibility and documentation.

Maintain a User Base through Utility and Ecosystem Placement. Across configurations, much of the work our interviewees described related to establishing a user base through technical means. By providing useful software and positioning themselves effectively within a software ecosystem, projects in our sample sought to grow their user base. However, users brought work: bugs, feature requests, work to integrate and maintain lower quality code contributions.

The developers we interviewed conceptualized useful software as code that was well documented, bug free, and that met functional and performance needs. They strategized to meet these goals because users justified and funded their continued work; having a large user base was thought of as a way to gain more grants. Collaborations, conferences, workshops, and surveys were leveraged to learn about user needs. We focus here on the labor of obtaining useful code contributions because of its importance for peer production, but it is notable that the efforts to demonstrate impact through high user numbers is another approach to creating and leveraging value for sustainability. Peer production communities and other configurations like labs, tool groups, and businesses all expected to obtain funding through serving users.

Some users contributed code to the software projects to meet a need of their own (a key feature of peer production sustainability), but project owners sometimes expressed misgivings about such user contributions. One PI, whose open source project presents as a lab, expressed that the SI2 program "drags [us] kicking and screaming into releasing the software for mass consumption" rather than technical merit, joking that they missed having "zero users, zero complaints." about their high performance, technically excellent software. Furthermore, highlighting the risks of sustainability through peer production, users' contributions were cited as possibly lowering the quality of the code, as an impetus for time consuming mentoring work, and as begetting more work because one feature might beg the introduction of another.

While middleware projects had a hard time assessing how many users they had, they benefited from that status all the same. Projects that were leveraged by other software packages as dependencies gained users. Several interviewees told us that these users were blind to their supporting infrastructure, a disadvantage for them when arguing for impact. However, one member of a peer production community described how his project reaped contributions from its status as middleware; developers from their downstream dependencies contributed code to the middleware project because of its use value to them. Others emphasized the benefits of upstream dependencies; one interviewee described theirs as "a gateway to a community of users."

Some software projects embedded themselves within the relevant ecosystem successfully by leveraging personal relationships with members of software projects. One interviewee said, "That's

important because we got buy-in from these individual experiments to this common toolkit." Interviewees maintained these relationships and the utility of their ecosystem by "taking care of the food chain," contributing patches to their upstream dependencies.

Pursue Funding. "Chasing money" was a concern for any long-term project, as was described by Bietz et al [9, 10]. Most often, interviewees and web presences indicated that project work was grant-funded. However, some projects generated revenue by selling licenses, creating a membership-based advisory committee, or establishing a business to charge for user support. Interviewees indicated that they find more opportunities for addressing "scientific problems" that can require new software features, but that they perceive less (although growing) support for maintenance work. Several projects in our sample benefited from parent organizations like national labs or federally funded organizations who provided funding that interviewees perceived as "more stable" and "longer term". In-kind donations from industry partners were resources for some software projects.

Insufficient funding for the required work prompted some developers to reorganize their software project from peer production to a business, but this pivot also required further changes: "That changed the skill set we hired, that changed the architecture of the service. So I have a user experience manager on my team ... [before] even product managers were not the on the previous team."

5.1.2 Sustainability Challenges for Peer Production. The investments described above that project leads make in pursuit of sustainability have uncertain returns. Developers might find their efforts entirely ineffective or that they receive too many extractive contributions. This is a risk of peer production in any environment, but this risk appears increased in academia because of the high barrier to entry.

We specifically asked our interviewees to estimate their total number of potential users and to estimate the number of people who could usefully contribute to their software. Estimates about potential user communities were consistently small (numbers around 100 were frequent) and the potential number of contributors within were much smaller (estimates in the 10s and 20s were frequent). The reality, as seen by our interviewees, was that it is rare to have sufficient enough knowledge of science and software development to make useful contributions—qualified contributors are a small and finite population. This is exacerbated by high contributor turnover. "We need a constant influx of new developers and contributors because old ones just drop out," said one interviewee.

While the challenge of converting users to contributors is common throughout open source, it is possible that the challenge is significantly more acute in academia. Not only are overall numbers low, but the socialization and training needed to make a possible contributor occurs during training for undergraduates and graduate students. This process involves using the software currently used in the lab—often the software that the PI was trained with, or perhaps created. This process creates strong path dependence in software, as students and graduates have deep investments in learning and working with particular pieces of software. The process also leads to networks of collaborators who are also trained and familiar with particular software. Combined, these factors ensure it is costly to switch software. Unlike areas with highly popular open projects, such as Ruby on Rails, the arrival rate of new and relatively uncommitted participants who can be attracted to novel software projects is likely much lower in academia.

An Alternative Approach. The contributors who do become involved in scientific peer production communities are made up of autonomous and might come "out of nowhere with their pull request" to the project. However, labs, tool groups, author groups, businesses, and consortia are not open to anyone. Contributors in these groups are usually hired or purposively selected. Through explicit or

social contract, project leads in these configurations ensure that they have control over the code and only enter into mentorship relationships that they anticipated.

The importance of this difference is made evident by labs' approaches to software sustainability. Some labs in our sample maintained software packages and toolkits for decades, sustaining their work in a closed environment that ensured their control over work and its outcomes. While converting unknown users into developers who make contributions is a defining feature of peer production sustainability, labs are characterized by their creation and leveraging of reputational value.

Labs attract the student labor that drives their software production. We saw many announcements on lab homepages about recent professional successes (e.g. publications, awards, graduations) and took this as evidence of how they attract such labor. PIs are able to plan a future for their work by finding a project for a student and assigning them to it. Students learn the necessary domain science and, possibly, software engineering skills under their PI's mentorship. All members of the lab increase their academic reputation as this relationship continues to produce code and subsequent publications. For students, the software project helps them establish a career. And, for PIs, the project maintains theirs. PIs use past productivity (made evident through publications and user bases) as warrant for future grant funding.

In an academic environment, project leads' reputation can be damaged by engaging in the obligatory sustainability practices of peer production. PIs succinctly described why they preferred not to engage in this kind of work: "I don't get any credit for this." In one case, a senior colleague made this especially clear to an interviewee, indicating that only certain publications were considered valuable:

> I had a discussion with whoever was head of department on the academic equality. And then the question came to the papers and he said, "Look. These papers, they are cited a lot, but it's just because people use your code."

Thus, our data showed that labs generate much of their reputation through work that excludes peer production maintenance and sustainability work. One developer told us, "You get your support, your grant, really for the scientific problems you're trying to solve—for the theory development."

Instead, labs create a skilled developer community around their software by helping students establish careers. Through the work of multiple generations of students or by hiring students as Research Software Engineers, labs progress their research and sustain their software. Furthermore, despite the reputational and career risks, long-term PI involvement ensures that at least one person is willing to do "the work that interests nobody".

5.1.3 Sustaining Other Software. As peer production does, labs, tool groups, author groups, businesses, and consortia upskill contributors, promote community, maintain user bases, and pursue funding. But, they apply different strategies and work toward different ends. The discussion above details how labs upskill students through mentorship that provides career and reputational benefit.

Alternative strategies are visible elsewhere as well. For example, some tool groups (who also selected their contributors) upskilled their workforce by ensuring the Research Software Engineers they hired had contact with any scientist developers. "There's been knowledge transfer from professional developers about good practices," one tool group member said.

Additionally, author groups, businesses, and consortia differed from peer production in their pursuit of funding. As temporary teams with scant web presences, author groups seemed to disband in favor of new collaborations for new grants awarded based on novelty. Businesses and consortia and consortia, however, obtained funding through sales and membership fees. The users and members who paid these fees then drove development; whereas labs exchanged reputation for funds, business and consortia traded use-value.

5.2 Transitioning to Peer Production

Transitioning to peer production involves overcoming any concerns, assuming the configuration's web genre, and adapting sustainability practices.

Overcoming Concerns. When reorganizing for peer production, project leads and contributors had to overcome a fear of being scooped or losing control of the project. One interviewee told us that before their open repository on Github, their project had a private repository to prevent their ideas from being stolen. " ...You always worry about being scooped as a scientist," they said. "...Especially if there's new and experimental...just making this publicly available, somebody who's in the know could see that and realize what you're trying to do and jump ahead of you." However, the small size of the community meant that few people would have the skills needed to scoop them. Thus, the team did openly release their code and they found their collaborator network and capabilities expanding: "Basically no one on the team had any expertise in [the contributor's area]....But this person apparently was extremely patient...It's not the world's the best [code]...but it worked...this would not have happened were we not open source."

Developers also feared that opening their code up to outside contributions would make them impotent to direct the project: "I actually remember now the kind of philosophy...which was a little bit anti-collaboration in terms of contribution...we're gonna lose control of the code, lose control of the architectures."

Hand-offs to peer production communities might also occur in the face of misgivings. A developer for a peer production project told us that they believe potential contributors fear losing credit for their work if it is integrated. To address this fear, their project includes a module to help users cite the components of the software that they used; by making the software work citable, they hoped to mitigate concern.

In the face of these concerns, some transitions might be more difficult to make than others. For instance, a reorganization from lab to peer production is especially unlikely (the only such case in our sample is described in 4.1.2 and appears to have begun years before) because they leverage their reputation for sustainability and many of the concerns listed above jeopardize that reputation.

Changing Web Presences. We saw that during a grant period, projects can change the configuration that they present as. While we originally anticipated the adoption of infrastructure like Github and continuous integration to be defining changes in a project transitioning to peer production, we found that all such software projects in our sample adopted these types of tools and they therefore lost most analytical power. Github is "the go to place for open source projects," said one interviewee, expressing a common sentiment.

However, adopting such open code infrastructure may be typical of most configuration styles in our sample, but only peer production projects also changed their online presence to establish open *communities*. Projects that reorganized to establish peer production also changed their websites to assume the configuration's online genre. They invited contributions and give instruction on how to do so, often by updating the README in their repositories. Projects that presented as peer production extended warm welcomes to their website visitors, explicitly emphasizing the project community.

Transitioning over Time. During our interviews we learned the early histories of software projects. These histories often included tales of transitions that occurred before the SI2 grant period (and, therefore, outside the scope of our content analysis), but some reached fruition during our period of observation. Some of the projects that maintained peer production during their SI2 grant had actually undergone apparent reorganizations in the years or decades prior. Other transitions were ongoing, moving between multiple configurations over time. An interviewee reflected on their

transition from a lab to a peer production project, describing a slow, organic process: "It's not unlike a tree growing. Ultimately, for major contributors...you can trace their path back to their origins in the [lab]...But, I think once the project becomes large enough, then people from the outside start contributing." We observed this project in the midst of this transition, starting the grant period as a tool group; their website had no invitation for outside contributions and the source code was not accessible. By the end of the grant period, they presented as peer production, corresponding to the interviewee's narrative. The project made their source code repository public and added information on how to contribute a PR to their code manual. Given the earlier history revealed through the interviews, we understand there to be decades of transition work that took the software from "restricted to the [lab] and collaborators" to having 100 contributors on Github.

Such examples showed that reorganization can take place over a long period of time. Hand-offs, in contrast, appeared to be readily accomplishable within the grant period. An exception in our data to the speed of hand-offs included a feature that took four years to hand-off from an author group to a peer production project. Described further in 4.1.2, the transition required extensive effort towards synergizing [9] that involved multiple pull requests, in-person meetings, and phone calls between the contributors and peer production developers. The hand-off interacted with its scientific setting when the author group's primary developer, a student, declared their need to move on: "Well, I need to graduate." Other hand-offs appeared to occur between more frequent collaborators, suggesting that some of the synergy work was accomplished through earlier interactions. Our rare examples of migration appeared to move swiftly as well; as described in 4.1.2, we were able to observe development work by the recipient organization during the grant period.

6 CONCLUSION

Through qualitative analysis of web content and interview data, we identified different organizational configurations for scientific software projects. We studied their approaches to transitioning their work to peer production communities as a sustainability strategy. We found that these transitions happen rarely during the grant period. Rather than a *transition* for peer production, the most frequent route to ending the grant period as peer production was to *continue* its practice from the start of the grant period. However, projects that made no transition still had to engage in sustainability work and the labor of scale [35]. Notable obstacles to transitioning include overcoming concerns about receiving academic credit and investing in peer production when its returns are uncertain and undervalued in science.

We learned that reorganization for peer production can occur over a long period of time but that hand-offs or migrations might move more quickly. A tempting implication for policy might be to suggest that, rather than encouraging reorganizations for peer production, funders focus on other modes of transition like hand-offs or migration. However, these options include costs of their own; synergizing work must be accomplished to ensure fruitful transitions and infrequent extractive contributions [29]. Furthermore, any transition to peer production will require the sustainability practices we have described here, sometimes referred to as "extra work" [76]. Our data, like prior research [39], show desire for academic credit is a strong disincentive to take extra work on. Responding to Jackson and colleague's call made to the CSCW community to inform science policy [42], we suggest that grant reviewers consider alignment between the contributing and receiving organizations. Have they worked together? Do the contributors demonstrate understanding of the recipient's code and conduct standards? Are there ecosystem affinities between the codes? Does the code fit in with the planned future for the project?

We have shown that the ability to recognize extractive contributions in advance is crucial to mitigating the risks of peer production. While open source development in all environments would benefit from such knowledge, we propose investigating this competence as an area of future

research because of the dual risks in academia: peer production sustainability work might fail, but even its success could have negative effects on scientists' careers and the hoped for benefits to science of sustainable, open software development.

ACKNOWLEDGMENTS

Thank you to Isabella Schloss for her help conducting and organizing interviews. Thanks also to the 2017 and 2018 Howison Lab research assistants who helped us refine our coding scheme.

This material is based upon work supported by the National Science Foundation under Grant No. 1453548.

REFERENCES

[1] Daniel Atkins. 2003. *Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure.* Technical Report. http://www.nsf.gov/od/oci/reports/toc.jsp

[2] Carliss Y. Baldwin and Kim B. Clark. 2006. The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model? *Management Science* 52, 7 (2006), 1116–1127.

[3] Evelyn J Barry, Chris F Kemerer, and Sandra A Slaughter. 1999. Toward a detailed classification scheme for software maintenance activities. In *Proceedings Of the 5th Americas Conference on Information Systems.*

[4] Christoph Becker, Stefanie Betz, Ruzanna Chitchyan, Leticia Duboc, Steve M. Easterbrook, Birgit Penzenstadler, Norbet Seyff, and Colin C. Venters. 2016. Requirements: The Key to Sustainability. *IEEE Software* 33, 1 (2016), 56–65.

[5] Yochai Benkler. 2002. Coase's Penguin, or, Linux and The Nature of the Firm. *Yale Law Journal* 112 (2002), 369–446.

[6] Yochai Benkler. 2017. Peer production, the commons, and the future of the firm. *Strategic Organization* 15, 2 (May 2017), 264–274. https://doi.org/10.1177/1476127016652606

[7] G. Bruce Berriman, John Good, Ewa Deelman, and Anastasia Alexov. 2011. Ten years of software sustainability at the Infrared Processing and Analysis Center. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 369, 1949 (Aug. 2011), 3384–3397. https://doi.org/10.1098/rsta.2011.0136

[8] Nikolai Bezroukov. 1999. Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism). *First Monday* 4, 10 (1999). http://firstmonday.org/issues/issue4_10/bezroukov/index.html

[9] Matthew J. Bietz, Eric P. Baumer, and Charlotte P. Lee. 2010. Synergizing in Cyberinfrastructure Development. *Computer Supported Cooperative Work* 19, 3-4 (Aug. 2010), 245–281. https://doi.org/10.1007/s10606-010-9114-y

[10] Matthew J. Bietz, Toni Ferro, and Charlotte P. Lee. 2012. Sustaining the development of cyberinfrastructure: an organization adapting to change. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work (CSCW '12).* ACM, New York, NY, USA, 901–910. https://doi.org/10.1145/2145204.2145339

[11] J Birnholtz. 2008. When Authorship Isn't Enough: Lessons from CERN on the Implications of Formal and Informal Credit Attribution Mechanisms in Collaborative Research. *Journal of Electronic Publishing* 11, 1 (2008).

[12] Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby. 1995. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of software engineering* 1, 1 (1995), 57–94.

[13] Geoffrey C. Bowker, Paul N. Edwards, Steven J. Jackson, and C. P. Knobel. 2010. The Long Now of Cyberinfrastructure. In *World Wide Research: Reshaping the Sciences and Humanities.* MIT Press, Cambridge, MA.

[14] Barry Bozeman. 2000. Technology transfer and public policy: a review of research and theory. *Research policy* 29, 4 (2000), 627–655.

[15] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[16] Jason Cory Brunson. 2020. ggalluvial: Layered Grammar for Alluvial Plots. *Journal of Open Source Software* 5, 49 (2020), 2017. https://doi.org/10.21105/joss.02017

[17] Jason Cory Brunson and Quentin D. Read. 2020. ggalluvial: Alluvial Plots in 'ggplot2'. http://corybrunson.github.io/ggalluvial/

[18] Johanna Cohoon and James Howison. 2021. Norms and Open Systems in Open Science. *Information & Culture* 56, 2 (July 2021), 115–137. https://doi.org/10.7560/IC56201

[19] Juliet Corbin and Anselm Strauss. 2014. *Basics of Qualitative Research.* SAGE. Google-Books-ID: Dc45DQAAQBAJ.

[20] Stephen Crouch, Neil Chue Hong, Simon Hettrick, Mike Jackson, Aleksandra Pawlik, Shoaib Sufi, Les Carr, David De Roure, Carole Goble, and Mark Parsons. 2013. The Software Sustainability Institute: Changing Research Software Attitudes and Practices. *Computing in Science Engineering* 15, 6 (Nov. 2013), 74–80. https://doi.org/10.1109/MCSE.2013.133 Conference Name: Computing in Science Engineering.

[21] Kevin Crowston, James Howison, and Hala Annabi. 2006. Information systems success in free and open source software development: Theory and measures. *Software Process: Improvement and Practice* 11, 2 (2006), 148. https:

//doi.org/10.1002/spip.259

[22] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2012. Free (Libre) Open Source Software Development: What We Know and What We Do Not Know. *Comput. Surveys* 44, 2 (2012), Article 7. https://doi.org/10.1145/2089125.2089127

[23] Jean-Michel Dalle and Guillaume Rousseau. 2004. Toward Collaborative Open-Source Technology Transfer. In *Proceedings of the ICSE 4th Workshop on Open Source*.

[24] Peter T. Darch and Ashley E. Sands. 2017. Uncertainty about the Long-Term: Digital Libraries, Astronomy Data, and Open Source Software. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. 1–4. https://doi.org/10.1109/JCDL.2017.7991584

[25] Ivan De Noni, Andrea Ganzaroli, and Luigi Orsi. 2013. The evolution of OSS governance: a dimensional comparative analysis. *Scandinavian Journal of Management* 29, 3 (Sept. 2013), 247–263. https://doi.org/10.1016/j.scaman.2012.10.003

[26] Michael Dorner, Maximilian Capraro, and Ann Barcomb. 2020. Quo Vadis, Open Source? The Limits of Open Source Growth. *arXiv:2008.07753 [cs]* (Oct. 2020). http://arxiv.org/abs/2008.07753 arXiv: 2008.07753.

[27] Robert R. Downs, W. Christopher Lenhardt, Erin Robinson, Ethan Davis, and Nicholas Weber. 2015. Community Recommendations for Sustainable Scientific Software. *Journal of Open Research Software* 3, 1 (Nov. 2015), e11. https://doi.org/10.5334/jors.bt Number: 1 Publisher: Ubiquity Press.

[28] Paul N Edwards. 2010. *A vast machine: Computer models, climate data, and the politics of global warming*. MIT Press, Cambridge, Mass.

[29] Nadia Eghbal. 2020. *Working in public: the making and maintenance of open source software / by Nadia Eghbal* (first edition ed.). Stripe Press, San Francisco, California. Book Title: Working in public : the making and maintenance of open source software.

[30] Roy T. Fielding. 1999. Shared leadership in the Apache project. *Commun. ACM* 42, 4 (April 1999), 42–43. https://doi.org/10.1145/299157.299167

[31] T. A. Finholt and Gary. M. Olson. 1997. From Laboratories to collaboratories: A new organizational form for scientific collaboration. *Psychological Science* 8, 1 (1997), 36.

[32] Peer C. Fiss. 2007. A Set-theoretic Approach to Organizational Configurations. *Academy of Management Review* 32 (2007), 1198.

[33] Brian Fitzgerald. 2006. The transformation of Open Source Software. *MIS Quarterly* 30, 3 (2006), 587–598.

[34] Karl Fogel. 2005. *Producing open source software: How to run a successful free software project.* " O'Reilly Media, Inc.".

[35] R. Stuart Geiger, Dorothy Howard, and Lilly Irani. 2021. The Labor of Maintaining and Scaling Free and Open-Source Software Projects. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (April 2021), 1–28. https://doi.org/10.1145/3449249

[36] Carole Goble, David De Roure, and Sean Bechhofer. 2013. Accelerating Scientists' Knowledge Turns. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, Ana Fred, Jan L. G. Dietz, Kecheng Liu, and Joaquim Filipe (Eds.). Number 348 in Communications in Computer and Information Science. Springer Berlin Heidelberg, 3–25.

[37] James Howison and Kevin Crowston. 2014. Collaboration through open superposition: A theory of the open source way. *MIS Quarterly* 38, 1 (2014), 29–50. https://doi.org/10.25300/MISQ/2014/38.1.02

[38] James Howison and James D. Herbsleb. 2011. Scientific software production: incentives and collaboration. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. Hangzhou, China, 513–522. https://doi.org/10.1145/1958824.1958904

[39] James Howison and James D. Herbsleb. 2013. Incentives and integration in scientific software production. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. San Antonio, TX, 459–470. https://doi.org/10.1145/2441776.2441828

[40] Xing Huang, Xianghua Ding, Charlotte P. Lee, Tun Lu, and Ning Gu. 2013. Meanings and Boundaries of Scientific Software Sharing. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work (CSCW '13)*. ACM, New York, NY, USA, 423–434. https://doi.org/10.1145/2441776.2441825

[41] Philipp Hukal, Nicholas Berente, Matt Germonprez, and Aaron Schecter. 2019. Bots Coordinating Work in Open Source Software Projects. *Computer* 52, 9 (Sept. 2019), 52–60. https://doi.org/10.1109/MC.2018.2885970 Conference Name: Computer.

[42] Steven J Jackson, Stephanie B Steinhardt, and Ayse Buyuktur. 2013. Why CSCW needs science policy (and vice versa). In *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 1113–1124.

[43] Caroline Jay, Robert Haines, Daniel S. Katz, Jeffrey C. Carver, Sandra Gesing, Steven R. Brandt, James Howison, Anshu Dubey, James C. Phillips, Hui Wan, and Matthew J. Turk. 2020. The challenges of theory-software translation. *F1000Research* 9 (Oct. 2020), 1192. https://doi.org/10.12688/f1000research.25561.1

[44] Daniel S. Katz, Sou-Cheng T. Choi, Hilmar Lapp, Ketan Maheshwari, Frank Löffler, Matthew Turk, Marcus D. Hanwell, Nancy Wilkins-Diehr, James Hetherington, James Howison, Shel Swenson, Gabrielle D. Allen, Anne C. Elster, Bruce Berriman, and Colin Venters. 2014. Summary of the First Workshop on Sustainable Software for Science: Practice and

Experiences (WSSSPE1). *Journal of Open Research Software* 2, 1 (July 2014), e6. https://doi.org/10.5334/jors.an arXiv: 1404.7414.

[45] Daniel S. Katz, Kenton McHenry, and Jong S. Lee. 2021. Research Software Sustainability: Lessons Learned at NCSA. In *Proceedings of the 54th Hawaii International Conference on System Sciences*. Manoa, HI. https://doi.org/10.24251/HICSS.2021.873 Accepted: 2020-12-24T20:30:52Z Pages: 7249.

[46] Daniel S. Katz, Lois Curfman McInnes, David E. Bernholdt, Abigail Cabunoc Mayes, Neil P. Chue Hong, Jonah Duckles, Sandra Gesing, Michael A. Heroux, Simon Hettrick, Rafael C. Jimenez, Marlon Pierce, Belinda Weaver, and Nancy Wilkins-Diehr. 2019. Community Organizations: Changing the Culture in Which Research Software Is Developed and Sustained. *Computing in Science Engineering* 21, 2 (March 2019), 8–24. https://doi.org/10.1109/MCSE.2018.2883051 Conference Name: Computing in Science Engineering.

[47] David J. Ketchen, James G. Combs, Craig J. Russell, Chris Shook, Michelle A. Dean, Janet Runge, Franz T. Lohrke, Stefanie E. Naumann, Dawn Ebe Haptonstahl, Robert Baker, Brenden A. Beckstein, Charles Handler, Heather Honig, and Stephen Lamoureux. 1997. Organizational Configurations and Performance: A Meta-Analysis. *The Academy of Management Journal* 40, 1 (1997), 223–240. https://doi.org/10.2307/257028 Publisher: Academy of Management.

[48] Karin Knorr-Cetina. 1999. *Epistemic Communities*. Harvard Education Press, Cambridge, MA.

[49] Bruno Latour and Steve Woolgar. 1979. *Laboratory Life: The Social Construction of Scientific Facts*.

[50] Charlotte P. Lee, Paul Dourish, and Gloria Mark. 2006. The human infrastructure of cyberinfrastructure. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work (CSCW '06)*. ACM, New York, NY, USA, 483–492. https://doi.org/10.1145/1180875.1180950

[51] Sheen S. Levine and Michael J. Prietula. 2013. Open Collaboration for Innovation: Principles and Performance. *Organization Science* 25, 5 (Dec. 2013), 1414–1433. https://doi.org/10.1287/orsc.2013.0872

[52] Eric T. Meyer and Ralph Schroeder. 2015. *Knowledge Machines: Digital Transformations of the Sciences and Humanities*. MIT Press, Cambridge, MA, USA.

[53] Henry Mintzberg. 1980. Structure in 5's: A Synthesis of the Research on Organization Design. *Management Science* 26, 3 (1980), 322–341. http://www.jstor.org/stable/2630506

[54] NSF. 2016. Software Infrastructure for Sustained Innovation (SI2: SSE & SSI). https://www.nsf.gov/pubs/2016/nsf16532/nsf16532.htm

[55] NSF. 2019. Transforming Science Through Cyberinfrastructure: NSF's Blueprint for a National Cyberinfrastructure Ecosystem for Science and Engineering in the 21st Century. (2019), 33.

[56] Gary M. Olson, Ann Zimmerman, Nathan Bos, and William Wulf. 2008. *Scientific Collaboration on the Internet*. The MIT Press, Cambridge, MA.

[57] Drew Paine and Charlotte P. Lee. 2020. Coordinative Entities: Forms of Organizing in Data Intensive Science. *Computer Supported Cooperative Work (CSCW)* 29, 3 (June 2020), 335–380. https://doi.org/10.1007/s10606-020-09372-2

[58] B. Penzenstadler, V. Bauer, C. Calero, and X. Franch. 2012. Sustainability in software engineering: A systematic literature review. In *16th International Conference on Evaluation Assessment in Software Engineering (EASE 2012)*. 32–41. https://doi.org/10.1049/ic.2012.0004

[59] Andreas Prlić and James B. Procter. 2012. Ten Simple Rules for the Open Development of Scientific Software. *PLOS Comput Biol* 8, 12 (Dec. 2012), e1002802. https://doi.org/10.1371/journal.pcbi.1002802

[60] Eric S. Raymond. 1998. The Cathedral and the Bazaar. *First Monday* 3, 3 (1998). http://www.firstmonday.org/issues/issue3_3/raymond/index.html

[61] David Ribes and Thomas A. Finholt. 2007. Planning infrastructure for the long-term: Learning from cases in the natural sciences. In *Proceedings of the Third International Conference on e-Social Science*. http://davidribes.com/docs/RibesFinholt-LearningFromEScience(submitted).pdf

[62] Jeffrey A. Roberts, Il-Horn Hann, and Sandra A. Slaughter. 2006. Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science* 52, 7 (2006), 999. https://doi.org/10.1287/mnsc.1060.0554

[63] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Martin Michlmayr. 2005. Evolution of Volunteer Participation in Libre Software Projects: Evidence from Debian. In *Proceedings of the First International Conference on Open Source Systems*, Marco Scotto and Giancarlo Succi (Eds.). 107.

[64] Charles M. Schweik and Robert C. English. 2012. *Internet Success: A Study of Open-Source Software Commons*. MIT Press. Google-Books-ID: 1tbxCwAAQBAJ.

[65] J. Segal. 2008. Models of Scientific Software Development. In *Proc. 2008 Workshop Software Eng. in Computational Science and Eng. (SecSe 08)*. http://www.cse.msstate.edu/~SECSE08/Papers/Segal.pdf

[66] S. K. Shah. 2006. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science* 52, 7 (2006), 1000–1014. https://doi.org/10.1287/mnsc.1060.0553

[67] Maha Shaikh and Ola Henfridsson. 2017. Governing open source software through coordination processes. *Information and Organization* 27, 2 (June 2017), 116–135. https://doi.org/10.1016/j.infoandorg.2017.04.001

[68] Steven Shapin. 2010. *The Scientific Life: A Moral History of a Late Modern Vocation* (illustrated edition ed.). University of Chicago Press, Chicago.

[69] Jeremy C. Short, G. Tyge Payne, and David J. Ketchen. 2008. Research on Organizational Configurations: Past Accomplishments and Future Challenges. *Journal of Management* 34, 6 (Dec. 2008), 1053–1079. https://doi.org/10.1177/0149206308324324

[70] Susan Leigh Star and Karen Ruhleder. 1994. Steps Towards an Ecology of Infrastructure: Complex Problems in Design and Access for Large-scale Collaborative Systems. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work (CSCW '94)*. ACM, New York, NY, USA, 253–264. https://doi.org/10.1145/192844.193021

[71] Paula Stephan. 2012. *How Economics Shapes Science* (1 ed.). Harvard University Press.

[72] Steven Weber. 2005. *The Success of Open Source — Steven Weber*. Harvard University Press, Boston, MA. https://www.hup.harvard.edu/catalog.php?isbn=9780674018587

[73] John Swales. 1990. *Genre analysis: English in academic and research settings*. Cambridge University Press, Cambridge [England] ; New York.

[74] David J Teece. 1986. Profiting from technological innovation: Implications for integration, collaboration, licensing and public policy. *Research policy* 15, 6 (1986), 285–305.

[75] Jonathan Tennant, Ritwik Agarwal, Ksenija Baždarić, David Brassard, Tom Crick, Daniel J. Dunleavy, Thomas Rhys Evans, Nicholas Gardner, Monica Gonzalez-Marquez, Daniel Graziotin, Bastian Greshake Tzovaras, Daniel Gunnarsson, Johanna Havemann, Mohammad Hosseini, Daniel S. Katz, Marcel Knöchelmann, Christopher R Madan, Paolo Manghi, Alberto Marocchino, Paola Masuzzo, Peter Murray-Rust, Sanjay Narayanaswamy, Gustav Nilsonne, Josmel Pacheco-Mendoza, Bart Penders, Olivier Pourret, Michael Rera, John Samuel, Tobias Steiner, Jadranka Stojanovski, Alejandro Uribe-Tirado, Rutger Vos, Simon Worthington, and Tal Yarkoni. 2020. *A tale of two 'opens': intersections between Free and Open Source Software and Open Scholarship*. preprint. SocArXiv. https://doi.org/10.31235/osf.io/2kxq8

[76] Erik H. Trainer, Chalalai Chaihirunkarn, Arun Kalyanasundaram, and James D. Herbsleb. 2014. Community Code Engagements: Summer of Code & Hackathons for Community Building in Scientific Software. In *Proceedings of the 18th International Conference on Supporting Group Work (GROUP '14)*. ACM, New York, NY, USA, 111–121. https://doi.org/10.1145/2660398.2660420

[77] Erik H. Trainer, Chalalai Chaihirunkarn, Arun Kalyanasundaram, and James D. Herbsleb. 2015. From Personal Tool to Community Resource: What's the Extra Work and Who Will Do It? ACM Press, 417–430. https://doi.org/10.1145/2675133.2675172

[78] Janet Vertesi. 2020. *Shaping Science: Organizations, Decisions, and Culture on NASA's Teams*. University of Chicago Press, Chicago, IL. https://press.uchicago.edu/ucp/books/book/chicago/S/bo49911700.html

[79] Eric von Hippel and Georg von Krogh. 2003. Open Source Software and the 'Private-Collective' Innovation Model: Issues for Organization Science. *Organization Science* 14, 2 (2003), 209–223.

[80] P. Wagstrom, James D. Herbsleb, R. E. Kraut, and A. Mockus. 2010. The Impact of Commercial Organizations on Volunteer Participation in an Online Community. http://www.casos.cs.cmu.edu/publications/papers/2010The%20ImpactOfCommericial.pdf

[81] Karl E. Weick. 1989. Theory construction as disciplined imagination. *Academy of Management Review* 14, 4 (1989), 516–531.

[82] Susan Winter, Nicholas Berente, James Howison, and Brian Butler. 2014. Beyond the organizational 'container': Conceptualizing 21st century sociotechnical work. *Information and Organization* 24, 4 (Oct. 2014), 250–269. https://doi.org/10.1016/j.infoandorg.2014.10.003

[83] Joanne Yates and Wanda J. Orlikowski. 1992. Genres of Organizational Communication: A Structurational Approach to Studying Communication and Media. *The Academy of Management Review* 17, 2 (1992), 299–326. https://doi.org/10.2307/258774