The Labor of Maintaining and Scaling Free and Open-Source Software Projects

R. STUART GEIGER*, University of California, San Diego; Department of Communication and Halicioglu Data Science Institute, USA
DOROTHY HOWARD, University of California, San Diego; Department of Communication and Feminist Labor Lab, USA
LILLY IRANI, University of California, San Diego; Department of Communication, The Design Lab, and Feminist Labor Lab, USA

Free and/or open-source software (or F/OSS) projects now play a major and dominant role in society, constituting critical digital infrastructure relied upon by companies, academics, non-profits, activists, and more. As F/OSS has become larger and more established, we investigate the labor of maintaining and sustaining those projects at various scales. We report findings from an interview-based study with contributors and maintainers working in a wide range of F/OSS projects. Maintainers of F/OSS projects do not just maintain software code in a more traditional software engineering understanding of the term: fixing bugs, patching security vulnerabilities, and updating dependencies. F/OSS maintainers also perform complex and often-invisible interpersonal and organizational work to keep their projects operating as active communities of users and contributors. We particularly focus on how this labor of maintaining and sustaining changes as projects and their software grow and scale across many dimensions. In understanding F/OSS to be as much about maintaining a communal project as it is maintaining software code, we discuss broadly applicable considerations for peer production communities and other socio-technical systems more broadly.

CCS Concepts: • **Social and professional topics** → **Computer supported cooperative work**; **Socio-technical systems**; **Computing profession**; **Project and people management**; • **Software and its engineering** → **Open source model**.

Additional Key Words and Phrases: open source, free software, maintenance, infrastructure, labor

ACM Reference Format:
R. Stuart Geiger, Dorothy Howard, and Lilly Irani. 2021. The Labor of Maintaining and Scaling Free and Open-Source Software Projects. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW1, Article 175 (April 2021), 28 pages. https://doi.org/10.1145/3449249 https://stuartgeiger.com/papers/maintaining-scaling-foss-cscw2021.pdf

1 INTRODUCTION

Free and/or open-source software (or F/OSS) refers to a broad set of working processes, social movements, and organizations that have formed around the production and distribution of software,

*The majority of the work on this project was conducted when Geiger was affiliated with the Berkeley Institute for Data Science at the University of California, Berkeley

Authors' addresses: R. Stuart Geiger, University of California, San Diego; Department of Communication and Halicioglu Data Science Institute, 9500 Gilman Dr, La Jolla, California, USA, 92093; Dorothy Howard, University of California, San Diego; Department of Communication and Feminist Labor Lab, 9500 Gilman Dr, La Jolla, California, USA, 92093; Lilly Irani, University of California, San Diego; Department of Communication, The Design Lab, and Feminist Labor Lab, 9500 Gilman Dr, La Jolla, California, USA, 92093.

2573-0142/2021/4-ART175
https://doi.org/10.1145/3449249

with a complex and contested history going back decades. These movements have been extensively studied from many disciplinary perspectives, as well as the subject of substantial commentary from its members, across its many factions [e.g. 49, 102, 125]. These software projects publicly release their source code, rather than various commercial models of software in which firms require payment to use software and/or restrict the ability for users to modify the software. Practitioners often describe F/OSS as being 'free' in two ways: free in being available at no cost (called "free as in beer"), and free in having source code available and licensed such that users can modify it (called "free as in speech") [78]. However, it is important to ask about how the work of maintaining these projects fits into these paradigms of free-ness, when F/OSS and other similar peer production projects require labor and material resources [41, 84, 124].

In prior decades, many early F/OSS projects began as hobbyist efforts to build alternatives to commercial proprietary software from the tech industry. Many early contributors volunteered spare time or negotiated with their employer to let them spend work time on F/OSS [8, 27, 29, 64, 76, 94]. Many F/OSS projects have become more commercial and part of the tech sector over the past two decades [10, 47]. Today, F/OSS has grown such that many projects have become the dominant product in their sector and are extensively relied upon by commercial software firms (e.g. Linux, Apache, Python). Many of the most successful F/OSS projects are not user-facing applications, but software infrastructure that are relied upon by companies inside and outside of the software industry, such as operating systems, programming languages, software libraries, servers, and web components. A 2020 survey of 950 enterprise-sized companies across sectors reported that 95% said open source software was important to their infrastructure strategy and 77% would adopt more open source software the next year [103]. F/OSS is also relied upon by government entities, non-profits, and activist movements, where the free cost and the ability to modify it can be crucial.

As F/OSS projects have become more critically embedded into organizations and economies, there has been a major shift about questions of "sustainability" within many projects, especially those that began as volunteers' side projects. This term is used to call attention to whether projects will keep developing and maintaining what others rely on, as all software must be maintained to continue to be useful to its users. Nadia Eghbal's influential report on the topic opens with the Heartbleed bug in OpenSSL, a F/OSS software library used by two-thirds of websites to handle encryption, leading to the worst security vulnerability in the web's history [41]. Despite the critical centrality of OpenSSL, the project's maintainers had long struggled to find the time and money to work on it. Eghbal quotes the lead maintainer's public post: "The mystery is not that a few overworked volunteers missed this bug; the mystery is why it hasn't happened more often" [88]. Eghbal's recent work suggests that small numbers of individual developers often do the bulk of the work in many F/OSS projects and can have somewhat transactional relationships with contributors and users, in contrast to how predominant narratives present F/OSS as composed of large collaboration-driven communities [42].

Our research question asks how the work of maintaining these projects changes as F/OSS projects become key dependencies for others, including well-resourced organizations in and out of the tech sector. We conducted 37 qualitative interviews with current or former F/OSS contributors and maintainers. Our focus was projects that began as purely-volunteer efforts and have since become widely relied upon as infrastructure for other organizations beyond the project. We find that as projects scale across all kinds of dimensions — number of users, contributors, or maintainers; kinds of users, contributors, or maintainers; size, complexity, and features of the codebase; interdependence in a software ecosystem; and more — the work of maintaining the project and the meaning of being a maintainer can dramatically change. Scale brings new tasks and changes the nature of existing tasks. For example, for projects with few or no users, providing technical support to users can be can be an exciting opportunity to grow the community around the project. Yet for a large-scale

project with millions of users, this can become an overwhelming flood of demands that requires establishing specific rules, roles, and norms, such as developing processes for triaging user support requests.

In particular, we find that the ostensibly-technical work of software engineering takes on more organizational, communicative, and even competitive aspects at larger scales. This is a well-established theme about the socio-technical nature of computer-supported cooperative work, of software engineering, and of work in general. However, our study details how the activities and experiences of this maintenance work change as projects grow, develop, and become embedded within broader networks of people, code, money, and institutions, including corporations, governments, academia, non-profits, and other F/OSS projects. We conclude by discussing the "scalar labor" of managing a project as it scales, such as how the deferral of this labor of scaling can create consequences for projects down the line – a problem we term "scalar debt." Maintenance tasks pile up, requiring a massive amount of often less-visible work to build more organizational capacity to keep up with an onslaught of demands. Finally, we discuss how F/OSS maintainers of popular projects also sometimes face additional work as they become hypervisible and even microcelebrities, which is contrary to how infrastructural maintenance is typically described as "behind the scenes" or "invisible" work.

2 BACKGROUND AND LITERATURE

2.1 Trajectories of F/OSS research

Our study took place in 2019-2020, during an era of F/OSS that is different to prior decades, when foundational works about F/OSS proliferated. These classic accounts (e.g. [24]) suggested that F/OSS is composed of ideologically-driven collaborative and voluntary communities, producing public goods intended to supplant proprietary software alternatives. Past work documented F/OSS's connections to early internet engineers and makers [16], universities and academic research, and an opposition to corporate-friendly copyright and patent law [25, 76]. Academics and practitioners have discussed F/OSS as a social movement, which has splintered, with "open source" rising as a competing movement to free software, one that transformed the original anti-commercial values of free software. [77].

Researchers have focused on how F/OSS contributors collaborate and organize. F/OSS projects that have an "open development model" [53] are studied with other "peer production" [6] communities like Wikipedia or citizen science. Unlike in firms where employees are directed by managers, these projects often rely on self-directed contributions from individuals or individuals working between private industry and voluntary F/OSS contributor communities. Popular accounts often marvel at the relatively high quality of products produced from this ostensibly 'anarchistic' approach (e.g. [123], see also critiques from [114, 124, 127]). However, past work has repeatedly shown how this is made possible though less-visible coordination, articulation, and conflict resolution work, done to review, assemble, and align others' contributions. [18, 31, 46, 69, 83, 124]. On leadership and F/OSS, more work involves predicting who becomes a leader [48, 59] or leaders' motivations [82], although past work has discussed the roles of leaders, who often resolve conflicts, mentor newcomers, set rules, and organize tasks [4, 30, 74].

In a literature review on F/OSS within and beyond Computer-Supported Cooperative Work (CSCW), Germonprez et al. [57] note that most studies in CSCW and Human-Computer Interaction (HCI) are about either "input" topics like developer motivations or "process" topics about collaboration and governance. They also detail the massive transformation in F/OSS from past decades. Early work often found that contributors were purely volunteers working in loosely-formalized quasi-organizations, operating more by ad-hoc rules. They cite more recent findings showing the

rise of paid roles [109], corporate involvement [10, 47, 56, 100], and more formal organizational structures [20, 45] — from non-profit foundations based on fundraising to revenue-generating business models — which are increasingly the norm, especially in popular and longstanding projects. Finally, Germonprez et al. note that F/OSS projects are often studied as single-project case studies — which [32] also find in their 2012 review. However, they discuss how contemporary F/OSS projects exist in "complex supply chains" [57, p. 9], with multiple cascading interdependencies with other F/OSS projects in meso-level ecosystems.

These supply chains are not only to other F/OSS projects. F/OSS projects have complex relationships with the tech industry, governments, academia, and non-profits. Ekbia and Nardi [43] discuss the wide dependency of industrial profit on volunteer or under-compensated labor, as part of a set of practices they call "heteromation." They argue the global economy relies extensively on digitally-managed forms of un- or under-compensated labor, from user-generated content to microtasking platforms to F/OSS. What has been variously called crowdsourcing [15], cognitive surplus [115], or peer production and the "wealth of networks" [6], Ekbia and Nardi argue can all be understood as heteromation. Computational industries that benefit from this labor are being subsidized by other institutions that support these people working cheaply or for free, ranging from welfare states, universities, family, or charity. While issues of money, financial sustainability, and corporate relationships have long been studied in F/OSS. what is less studied are the lived experiences of F/OSS maintainers as their projects become enmeshed within institutions that can have significant access to financial, social, or cultural capital.

2.2 Infrastructural maintenance labor

Scholars have long drawn attention to how the work of maintaining technologies is often ignored and neglected. This scholarship notes the importance of maintenance in shaping the forms and functions of technologies beyond moments of invention [39, 111]. Scholars in Computer-Supported Cooperative Work have long emphasized how less visible, ostensibly 'non-technical' labor is crucial to the functioning of computational infrastructures, especially the "human infrastructure" in scientific cyberinfrastructure [80]. This work is often underbudgeted because it is unrecognized or undervalued, but necessary to make those systems 'seamless' [107] to their users. Eghbal draws more on metaphors of public infrastructure such as road and other public works to suggests that F/OSS is infrastructure needs to be considered through that lens [41].

Following Jackson's call to take maintenance and repair as the essence of technology [73], empirical studies of such practices have become more common in many areas. One theme is that tasks and roles construed as "maintenance" or "repair" often involve responsibilities beyond the technological, particularly in maintaining and repairing social and institutional relationships [37, 66, 71, 72, 110]. Infrastructure and maintenance work is also often discussed alongside other work sometimes invisibilized because of gendered or classed assumptions about its nature and importance [44, 51, 70, 117].[1] For example, Orr's ethnography of photocopy repair technicians showed how they can serve as the customer's primary point of contact with the photocopy company, managing that relationship more than designated account representatives [99]. Suchman's analyses of expert systems emerging from Xerox PARC also found that computer engineers underestimated the complexity of what secretaries do [121]. However, few in this literature have examined cases in which maintenance work becomes highly visible and even constitutive of leadership, as it can be in large-scale F/OSS projects.

[1]While there is often an impulse to always make work visible, other literature show how making this work more visible can come with regimes of surveillance, micromanaging, or self-censorship [14, 97, 120].

2.3 The many meanings of "scale"

We are interested in how the work of maintaining F/OSS projects changes as projects scale, but what exactly is scale? In and out of F/OSS, it is common to refer to organizations, communities, and platforms as being "small scale" or "large scale," which often compresses many aspects into a single term. In CSCW and HCI, scale is often a synonym for number of users, where classic work designed systems intended to operate with a pre-defined range of simultaneous users [e.g. 61]. In anthropology, Carr and Lempert [122] argue that when people use terms like "large scale " or "at scale," they often are intuiting a kind of synthetic construct that combines multiple related but distinct measures; in our case, like number of users, number of user organizations, kinds of users, interdependence in an ecosystem, number of contributors, and so on.

Recent work in CSCW has similarly identified more multivalent understandings of scale. These include Lee & Paine's "model of coordinated action" [81], in which they identify seven different dimensions along which software-mediated organizations can range: number of participants, number of communities of practice, physical/geographical distribution, nascence of routines, planned permanence, rate of turnover, and level of a/synchronicity in interactions. Our findings relate to both how these specific dimensions emerged as relevant for F/OSS maintainers, as well as their insight that shifts in each these dimensions can occur independently, but shifts in one dimension can also impact or depend on all the others. Studies of scientific cyberinfrastructures more widely demonstrate this theme, where scaling also includes integrating with interdependent projects and standards [3] – often called "embeddedness" [9, 40, 117] — and supporting a wider range of use cases over longer periods of time [75]. These studies have shown the various "tensions across the scales" [105] that emerge as projects grow.

3 METHODS AND METHODOLOGY

3.1 Research methods

This qualitative research is primarily based on semi-structured interviews with 37 maintainers of F/OSS projects in 2019-2020. Interviews lasted a median of 55 minutes and covered a range of open-ended topics, including top-level questions on: the interviewee's personal history in F/OSS, the kinds of work they do, how their roles and participation in the project has changed over time, governance and decision-making, funding and financial sustainability, motivation/demotivation and burnout, careers, how technologies and platforms impact participation, and work/life balance.

As is common with non-random sampling in qualitative research, we sought to strategically sample for diversity across many dimensions [95], rather than seek the kind of random uniform representative sample common in survey research. We specifically choose to recruit and interview a broad set of maintainers that varied across geography, national origin, age, employment status and sector, and gender. We made these efforts to sample for demographic diversity, while reflecting on how structural problems such as the gender gap among F/OSS contributors [38] present challenges to recruiting a diverse sample. We did not originally ask interviewees their demographics, but sent a post-interview survey, which 85% completed. For gender, 19% identified as women/female, 81% as men/male, and 0% as non-binary or other. For race/ethnicity (which allowed multiple selections), 72% identified as white/Caucasian (66% exclusively so), 16% as Hispanic/Latinx, 13% as Indian/South Asian, 6% as East or Southeast Asian, 3% as Black/African, and 3% as other. Interviewees were born in 14 different countries on 5 continents; the US was the most common with 47%. Interviewees currently reside in 12 different countries on 5 continents; the US was the most common with 56%. Ages spanned 25 to 64 years old, with 53% aged 30-39 years old.

We also sampled for diversity to recruit maintainers of different kinds of F/OSS projects. The projects whose maintainers we interviewed range from having a single developer-maintainer to

hundreds of contributors, and have a similar variance in terms of number of users. Some have existed for decades and have complex governance structures, roles, and norms, while others are relatively new. These projects represent a range of topical areas and locations within the technical stack, including operating systems, programming languages, software libraries, development environments, web frameworks, servers, databases, packaging, data analytics and research computing, devops, electronic arts and media, and more. Our focus on ensuring that our interview pool included maintainers from projects across these dimensions of scale follows from existing work on the importance of scale as a component of ethnographic work on CSCW infrastructures [101, 104] and broader globally-distributed phenomena [86].

Our recruitment methods involved utilizing our existing personal networks, attending F/OSS conferences and events, a call for participation shared on Twitter, and cold e-mailing F/OSS maintainers. We also conducted snowball sampling, asking our interviewees to suggest potential interviewees to us. To help our sampling for diversity, we utilized techniques similar to "trace ethnography" [55] in our recruitment methods to identify potential maintainers to recruit, based on available user data on social coding platforms including GitHub (see [34, 36, 126]. We identified core contributors through GitHub timelines, recent commits, and release notes. Our interviewees generally self-identified as current or former maintainers on their own terms. We interviewed maintainers who hold various roles within a wide range of F/OSS projects. We particularly focused on projects that have become relied upon as infrastructure by others and either are or began as largely based on volunteer labor. However, as we asked maintainers about all the projects they had worked on, we encountered projects beyond this.

3.2 Methodology and interpretive approaches

Our interpretive approach is grounded in symbolic interactionism, which focuses on how actors organize their interactions with the world, with one another, and with themselves through categories that emerge in their social worlds [12] and in wider discourses [22]. We have transcribed and inductively analyzed the interviews for themes using a grounded theory approach [23, 119], which involves a multi-stage process of coding statements with iteratively-generated themes. These themes identify social processes in common across our organizational site, generalizing across specific, local experiences while remaining bound to the particularities of the cultures and work processes under examination.

As we conducted interviews, many participants reflected not just on their own practices, but on the practices of others, including their broader theories about the political economy and history of F/OSS [67]. As we move from research to findings, we reflect upon how participants may have related information to researchers based on what they understood the study was about and who the research might affect, and reflexivity as a "recursive" [76] pattern in F/OSS communities [68]. Through member-checking in the form of sharing transcripts and our findings, we gave participants opportunities to give feedback and engage with our interpretations.

Our relationship to these communities is neither purely as outsiders or insiders. Our project was funded by non-profit foundations who also directly fund F/OSS projects. We have all worked as former or current F/OSS contributors or maintainers and have regularly attended various F/OSS-related meetups and events. All the authors are ethnographers embedded in larger ongoing research projects in this area — either in F/OSS projects themselves or in organizations that rely on and/or contribute to F/OSS projects. The data we present in this paper is centered on our set of 37 interviews, although our broader ethnographic experiences have informed both the kinds of questions we asked and how we interpreted interviewees' responses.

4 FINDINGS

4.1 What is a F/OSS maintainer and how does it change as projects scale?

"Maintainer" has a meaning within F/OSS that it rarely has in other technical domains. F/OSS maintainers do perform upkeep and repair work, but the term also usually connotes a leadership role, as [42] also discusses. This leadership role is often enacted through access permissions to the project's code repositories: becoming a maintainer typically involves being given the technical capacity to make changes to the project's code. This includes the capacity to approve or reject proposed changes (called "pull requests" in GitHub-style platforms) from non-maintainer contributors, as well as the capacity to moderate the issue trackers where anyone can report a bug or request a feature. Beyond this use of access permissions to formalize maintainer status, the role of a maintainer (and the use of this term) varied widely, which [42] also finds. Like in firms, organizations, and social movements, F/OSS projects range widely in size, scale, complexity, popularity, and interdependence. This makes it difficult and unwise to make overarching generalizations. We instead illustrate how maintainership differs across different kinds of projects, particularly focusing on how the labor of maintaining a F/OSS project changes as projects develop, grow, and scale.

In projects we encountered with fewer contributors, there was usually one individual in a more singular leadership role, who leads by doing a majority of the work. The most common term interviewees used to refer to such an individual was "the maintainer," although interviewees in corporate or academic institutions also noted that they sometimes code switch and use titles like "project lead" or "project manager" depending on the environment. In projects with a larger number of contributors, multiple maintainers often shared these responsibilities, sometimes with formal divisions of labor. In such projects, the leadership aspect of being what is sometimes called a "core maintainer" is analogous to being on a steering committee where decisions are made by consensus or voting [112].

However, even in projects with many maintainers, there was often a primary leadership role, typically held by the original creator or someone who took responsibility after the creator departed. One common term for this is the "benevolent dictator for life" or "BDFL," although one maintainer with such a role we interviewed described this more as "the person that tends to feel the most ownership for things that go wrong." Finally, while most projects we encountered used "maintainer" to describe these roles, some instead used "core developer" to signify this dual upkeep-leadership role, as [42] also discusses.

In the following sections, we identify kinds of tasks that are involved in F/OSS maintainer positions that change as the project grows in interdependencies, complexity, and users. Some of these only become apparent and necessary at certain scales, such as organizing events or coordinating with other F/OSS projects. Other tasks occur at all scales, but can become quite different at larger scales, such as providing support to users, fixing bugs, and developing new features. We do not intend this to be a comprehensive survey of maintenance work in F/OSS, and so focus on the relationship between scale and labor.

4.2 Maintaining Users: User support

When we asked our interviewees about the work of maintaining their projects, user support was a major topic. For the maintainer of a new project with few users, the first user who asks for help or raises an issue is a sign of validation and success. F/OSS projects are meant to be used, and a common attitude from maintainers of smaller projects was that whether the user's issue was due to their misunderstanding or a bug in the software, the maintainer can learn something from it. Maintainers told us how some users who ask for help become contributors and co-maintainers, or alternatively donors and patrons. Yet our interviewees who maintained large, well-known projects

with many users identified user support as an overwhelming and never-ending chore, particularly for projects that use GitHub-style collaboration platforms. One interviewee stated that "user support is something I do very regularly during the evenings during the week, or during the weekends. That actually takes a large chunk of my free time." For these maintainers, user support is an around the clock reality of their position.

Requests for user support come through many channels, including messages sent to their private e-mails and social media accounts. While users often seek software help in Q&A sites like StackOverflow, a project's maintainers are not generally obligated to be present in those spaces, although some do [134]. GitHub has allowed any user on the web with an account to open an issue for a F/OSS project hosted on that site. The number of open issues, potentially numbering in the thousands, is prominently displayed on the project's landing page, creating reputational pressure for the project. Managing and triaging the issue queue was often identified by our interviewees as a major task for maintainers of large-scale projects, although there was variation in the level of obligation. Some stated that maintainers may not have an obligation to actually fulfill the requests in the issue, but did have an obligation to respond and acknowledge the issue in a timely manner – sometimes described as within 24 or 48 hours, though others said that a week or more was acceptable.

Eghbal suggests maintainers engage in the "curation" of user and contributor interactions under intense time pressure [42]. As projects grew, larger projects often implement rules, recommendations, and even templates for raising issues. For example, it is common for larger projects to actively discourage using issues to request assistance in using properly-functioning parts of the software. However, a common tension arises when users report what they experience as bugs, but what the project's contributors and maintainers see this as the software operating properly. Maintainers of large or central projects we interviewed told us about users who were "disrespectful," "entitled," or "demanding," of their time and attention. This is also a growing topic of public discussion within F/OSS, with several talks and articles about how to be respectful to maintainers [19, 62, 79, 113].

As the work of investigating, triaging, and resolving issues intensified, maintainers reported feelings of demotivation, exhaustion, and burnout. This was especially common for maintainers of larger projects, who often mentioned user support as one of the more emotionally-intensive aspects of being a maintainer. As one interviewee discussed: "I think burnout can come from a lot of different things. It can come from constant bombardment of issues and notifications and you're constantly reminded of the things that you're not doing." Several interviewees noted that the way users interact with contributors and maintainers was crucial, with a few kind words or less-demanding phrasing going a long way for maintainers, which [131] also finds. However, this can be complicated in the global landscape of F/OSS, with several interviewees discussing cross-cultural or language barriers.

As projects scaled, interviewees described how reciprocity affected how they felt about F/OSS work. One interviewee stated that a top priority for them would be when the user requesting support is another F/OSS contributor in a related project seeking to fix a genuine bug that affects the ability for the two projects to be used together. Interviewees also expressed enthusiasm for supporting educational institutions, if an educator was having issues as part of teaching the software in their class. In contrast, maintainers we interviewed expressed frustration at the user support work generated when a large tech company integrated the F/OSS project as part of software they were building and selling — particularly if this company had not "given back" through financial donations or in-kind donations of labor (e.g. their developers regularly contributing to F/OSS projects). One interviewee noted that demanding free-riders are not limited to for-profit corporations, as some academic researchers had behaved with similar attitudes. The difference between how maintainers framed their experiences as collective work or exploited labor related to

the social and communicative relationships maintainers had to those with whom they collaborated, coordinated, or contributed.

4.3 Maintaining "Mainline" Code, Scaling Trust

F/OSS maintenance is not only about repairing and fixing. It is crucially about updating and changing to stay relevant. As the project grows, users expect a canonical version of the software, even as the number and diversity of contributors and user needs might expand. As contributors scale, maintainers must devise ways to scale trust. Version control practices are central to managing changes, especially with many contributors. The open development model of contemporary F/OSS projects typically involves a contributor making their own copy of the entire codebase, making whatever changes they see fit, and then submitting their modified version for review, approval, and merging. Traditionally, a maintainer decides what patches to accept and keeps a canonical version of the source code, regularly making public releases to keep the rest of the project up to date.

Smaller projects typically begin with a single maintainer, but as they begin to get more and more proposed contributions, some solo maintainers give a regular contributor commit rights, maintainer status, and let them manage specific releases. However, the founding maintainer must trust this new maintainer, because by default, they have the full technical privileges to accept any proposed changes. In one case mentioned in our interviews, a solo maintainer had been unable to spend as much time maintaining a project. When someone they did not know asked to be a co-maintainer, they happily accepted. However, this new maintainer added code to the software that silently used users' computers to mine cryptocurrency and deposit the profits in their account.

As projects scale the number of maintainers, code review processes are a common way of producing trustworthy code. Code review is the process by which one or more designated individuals who did not author the change have to approve a pull request before a maintainer can accept and merge it. The process is somewhat similar to academic peer review, especially in that there can be many cycles of review and revision between the code reviewer(s) and the original author. Code reviewers typically read through each line of code for specific issues, with contemporary social coding platforms supporting fine-grained line-level comments. Code reviewers typically look for bugs and inefficiencies, plus conformity with the project's code style, naming conventions, and approach to modularity. In some projects, only maintainers can do code review, but others allow a wider set of trusted non-maintainers to participate.

In smaller projects, code reviews might be informal and implicit, but formally specifying such rules can be a crucial aspect of scaling a project's number of contributors, code reviewers, maintainers, and codebase. As projects grow, maintainers have to devise ways distributing the work of review so they do not have to recheck submissions to the mainline or canonical version. For example, the Linux kernel still uses this mode of development, in which only its creator and lead maintainer Linus Torvalds can accept patches to the official "mainline" codebase. This was much easier when the project was much smaller, but it has since grown to many thousands of contributors. The project has developed a cascading "chain of trust" [28] where a top tier of subsystem maintainers are responsible for various sections of the codebase, making the decisions about what patches to accept. Some subsystem maintainers delegate responsibility further or have their own processes for making decisions about their part of the codebase. As the Linux kernel's documentation describes:

> "...top-level maintainers will ask Linus to 'pull' the patches they have selected for merging from their repositories. If Linus agrees, the stream of patches will flow up into his repository, becoming part of the mainline kernel. The amount of attention that Linus pays to specific patches received in a pull operation varies. It is clear that,

> sometimes, he looks quite closely. But, as a general rule, Linus trusts the subsystem maintainers to not send bad patches upstream." [28]

Version control and code reviews may seem purely technical, but they express the direction of the project. Authority to merge changes often means authority to set and enforce a specific vision of the project. Because of the necessity of keeping a single canonical code repository in this more traditional approach, the model of having a single lead maintainer who is ultimately the final decision-maker became widely prevalent in F/OSS, known as the "benevolent dictator" or "benevolent dictator for life" (BDFL). In one interview, a maintainer described a tense environment caused by the transition from a BDFL model to a more democratized system of decision-making. In essence, interpersonal relationships were strained when maintainers sought to democratize leadership roles within their project, after contributors felt their own speed and progress was limited by the power of the BDFL to veto group decisions and the BDFL resisted change.

4.4 The Labor of Managing and Maintaining Donations of Labor

Those who contribute code to F/OSS projects are donating the products of their labor, but those donations also generate new work for maintainers. When discussing the subject of what changes to merge, maintainers of projects with more contributors told us about perceived mismatches in expectations from non-maintainers who made the proposed changes / pull requests. In these cases, a non-maintainer added or expanded the code in a way they found useful. They contributed this code through a pull request, feeling they have generously donated time, effort, and intellectual property. However, from the maintainer's perspective, the new pull requests were a heavy obligation, both in the time to review them and the long-term costs in maintaining this code indefinitely. Wiggins [132] discusses a similar trend in citizen science as "free as in puppies," where such donations commit the recipient to care for it for years, even if the value of the contribution is uncertain.

As such, our interviewees who maintained F/OSS projects with more contributors mentioned the importance of rules requiring that new code follow certain standards that make the code easier to review and maintain. Maintainers we interviewed from projects with more rules and procedures around merging changes told us about cases in which the contributor became increasingly frustrated with what the maintainer was asking them to do in order to approve the proposed changes. In some cases, the contributor abandoned their contribution and the project altogether. In addition, sometimes the pull request perfectly conforms with all rules, but would take the project in a direction that the maintainers have decided is out of scope, and is thus rejected:

> "As a maintainer, you don't just merge things. You also try to be a thought leader for what's happening and why you're going to go a certain direction or not — and being able to politely say 'We don't want to go that way' for things you know you don't want to. That's the hardest thing, I think, because you can have situations where someone is earnestly trying to add something and you don't want to shut them down, but sometimes it's something that was declared you weren't going to do in this project..."

Because of the open contribution model in which anyone on the web can propose new changes, the work of being a maintainer of a F/OSS project typically involves a substantial amount of labor in managing the labor and even emotions of others. For our interviewees, this emotional labor was one of the more difficult and draining aspects of their position.

4.5 Scaling through automation: continuous integration, build systems, and testing

As projects grew in contributors, "continuous integration" (CI), build systems, and code testing were ways of automating code review and testing. CI involves automated processes that build a project's codebase across multiple platforms, then run pre-written, scripted tests to check if it is functioning

properly. Automated "linting" practices even check for proper formatting and style conventions within projects. CI services are directly integrated into GitHub-style platforms, which is one of the many ways bots and software automation can govern and transform virtual organizations [54, 107]. A 2016 survey found that 40% of the 34,000 most popular F/OSS projects on GitHub use CI, rising to 70% when examining the 500 most popular projects. [63]

The number of CI tests that are run can be staggeringly large. In the Python programming language's standard math library, there are currently 134 different "unit tests" that check the inputs and outputs of the square root (sqrt) function alone.[2] Major software libraries for programming languages can have tens of thousands of tests, and programming languages themselves can have hundreds of thousands of tests. These can be computationally intensive, a point we will return to.

For many maintainers of projects at all scales, continuous integration, build systems, and testing was a major strategy they relied on to automate the labor-intensive and interpersonally-intensive tasks of code review. This was particularly the case in rejecting code requests:

> "The other thing we've found is that if the computer tells them it's wrong, they take that better than if a human does. So, the more automated things you can do that will catch low-hanging fruit, the less offense it causes people. So [the] computer says, 'you've got a tab instead of spaces here,' they don't mind that. But if someone tells them that, they get grumpy about it."

Maintainers also described CI and testing as a strategy they relied upon to help them manage their workloads. In several interviews, when we asked about avoiding burnout, work/life balance, or advice to give to new maintainers, these strategies were the first responses. One interviewee who helps manage a large ecosystem of projects discussed how they require these kinds of measures for all projects in that ecosystem, such that "there are packages that I maintain that I have not updated in two plus years, because things just work. When something breaks, I will get an email."

Like all automation, these strategies redistribute and generate new forms of labor. Tests must continually be written and updated, especially when new features are added. Some projects require that new functions or features cannot be added without also adding appropriate levels of testing. Those gifts that create labor could be mitigated when they come with testing. As projects become increasingly complex, however, new kinds of *integration tests* must be written to check that the various subsets of the codebase work together. Testing also grows as projects become more integrated within an interdependent ecosystem of projects, all depending and relying on each other. It is increasingly common for projects to test that proposed changes do not break anything in other projects in the ecosystem. Although developing and maintaining these CI processes is itself a labor-intensive process, the work can be distributed to contributors who create automated tests for the code they write, rather than code reviewers responsible for catching bugs. In this way, it can help distribute maintainer labor out more widely as the project scales in terms of its codebase, featureset, contributor base, and interdependence within a software ecosystem.

Yet automation is computationally expensive, which can challenge F/OSS values that privilege relying on free and open platforms. A 2017 blog post about testing in the Rust programming language [2] reported that over 126,000 total tests are run for each proposed change/pull request across 20 different software configurations, with each taking 2 hours of computing time. The post references that the project only has resources to run one or possibly a few tests at the same time. Each additional pull request adds to the queue, meaning that contributors may have to wait for days to see if their proposed change breaks the testing suite. The author describes how longer queues can lead to conflict between contributors, who all need the CI system to approve their changes before they move on to the next stage of code review and approval.

[2] https://github.com/python/cpython/blob/master/Lib/test/cmath_testcases.txt

The only way for CI in some of these projects to scale up is to use either commercial cloud computing or a self-hosted server cluster. Contributors are supposed to run the full test suite on their own computer before submitting pull requests, but it is important for the project to test it in a wide range of configurations, as well as have a common public infrastructure that verifies the tests actually passed. GNU GCC, part of the free software movement, maintains their own distributed "compile farm," with donated servers hosted by those in the movement.[3] Commercial CI platforms have been growing, including Microsoft's Azure Pipelines (which runs on Microsoft's cloud computing infrastructure) and services from the venture capital funded companies CircleCI and AppVeyor (which run on cloud computing infrastructure from Google, Amazon, or Microsoft). These commercial CI platforms often give public F/OSS projects a free single CPU to run a single test at a time (Microsoft Azure currently gives 10 free simultaneous tests to F/OSS projects), but charge for more simultaneous tests – a necessity as projects grow in complexity. These commercial CI infrastructures challenge open source cultures that privilege creating autonomous, freely available infrastructures using freely available infrastructures, what anthropologist Chris Kelty has called a "recursive public" [76].

The decision to go beyond the free tier of CI services can be the first time a F/OSS project takes on a recurring financial expense, driving organizational changes and work. For smaller projects, free tiers are often sufficient. As projects grow their codebase and contributor base, they have to decide whether to fundraise to pay for more simultaneous tests or to deal with the strains of not being able to easily verify that proposed changes do not break the project. For those who fundraise for more CI resource, this can require projects to add accounting and financial roles (although events are more often the first time this occurs, which we discuss in a later section). Even the non-commercial, self-hosted alternative that projects like GNU GCC follow also require dedicated maintenance roles and the soliciting of donations to fund the self-hosted compile farm.

4.6 Ecosystem work: from interdependence to competition

One major dimension in which F/OSS projects scale is their interdependence on other F/OSS projects, which can take a variety of forms. First, they can become relied upon as critical infrastructure by other F/OSS projects. This is especially the case for software libraries, programming languages, and operating systems. Here, the number of users typically means the number of other developers building software using the F/OSS project. The chain of cascading dependencies can grow quite complex: a program may only rely on a few explicitly imported dependencies, but those projects and their dependencies can make the chain hundreds of projects long.

Maintainers must manage relationships with projects that are both "upsteam" (that they depend on) and "downstream" (that depend on them). If a project modifies its features, it can break downstream projects that are expecting this feature to work consistently. This is one common use for continuous integration, in which a project will regularly test its own functionality on beta or release versions of upstream projects it relies on. There are many issues that arise in interdependent software ecosystems beyond these more fine-grained issues around compatibility between new versions of software. One maintainer shared the difficulties that arose when a project they depended on began to face internal conflict, which forced them to either pick a side or do additional work to maintain compatibility with two projects.

Complex and interdependent F/OSS software ecosystems often find themselves needing to coordinate on high-level tasks and decisions. Conferences and conventions can be a major site to do this work. Some interviewees even described these ecosystem-level conferences in similar terms as political delegations, such as referring to a perceived need to send representatives. Some

[3] https://cfarm.tetaneutral.net/

have large blocks of time dedicated to open discussions on topics relevant to projects across the ecosystem. Such ecosystem-level topics include more software-specific issues that would require consensus to implement new features across an ecosystem, such as issues in such as packaging and release managers, data types, hardware support, or user telemetry.

Another major, perennial, and often controversial ecosystem-level topic is the proposed consolidation of related or competing projects within the same ecosystem. It is often common in F/OSS ecosystems for many projects to be created that solve a similar problem. There may be good reasons for multiple related competing projects to circulate in the same ecosystem, but navigating a crowded ecosystem can be confusing and frustrating for both users and developers. In such ecosystems, it can be common for someone to suggest there are too many competing packages and there needs to be a consolidation.

One interviewee shared a case where, during an open discussion session at an ecosystem-level conference, the maintainer of one project declared that some of the other competing projects in this ecosystem "needed to die," seeking to gain support for consolidation then and there. No consensus was reached, but it raises the issue that high-intensity communicative work and representation at such meetings becomes a solution to some vying to keep projects alive within an ecosystem, and to secure funds. A project's maintainers have can certainly decide to keep their project operating in an ecosystem that has decided to consolidate around another competing project, but they will find themselves with fewer and fewer users.

4.7 Growing a Community by Evangelizing

At a developer conference we attended for a particular subset of a F/OSS ecosystem, we observed a dedicated plenary session for 2-4 minute lightning talks, which were almost all pitches for a F/OSS project the speaker had developed. Many projects pitched were newer, less-established, and had not secured spots at the competitive conference, which some speakers noted. Maintainers asked for others to use their projects, with one explicitly imploring the audience to "rely upon" their project: to integrate it into their workflows and F/OSS projects. We asked about the role of this ritual: Why is it important to have a space for maintainers to convince others to use their often-fledgling F/OSS projects? And why did so many of them take the form of a "rely upon me" pitch, particularly when their project was not quite yet fully developed?

Our interviewees called these "rely upon us" pitches "evangelizing" and they brought it up frequently, in response to questions about scaling or even about general strategies for maintaining a project. In computing in general and F/OSS specifically, evangelizing is widely-used to describe efforts to sustain and maintain projects by bringing in more people [1, 85]. Maintainers repeatedly told us how important it was to have fellow contributors and maintainers to distribute the work and make their project sustainable. A key rationale was that users who rely on projects presumably become invested in those same projects' success — particularly when those users are also F/OSS developers or well-resourced organizations. When a project is used and relied upon by programmers and software firms, it gains access to a potential pool of skilled labor and resources. More contributors and users also make projects appear more successful [5], and thus worthy of funding by entities who fund F/OSS. Like many startups, projects often signal their credibility through showing the logos of well-known companies and universities that rely on their projects on their websites.

The tasks which constitute this vast domain of evangelizing can include: developing social media accounts for F/OSS projects, maintaining educational resources and documentation, building and updating websites with F/OSS project information, moderating and building Q&A sites and forums, and giving talks at meetups, conferences, companies, and schools. The promotional communicational, educational, and evangelical activities done by maintainers F/OSS lay the conditions for

further expansion and infrastructural change — and more maintenance work. Some interviewees said they deeply enjoyed evangelizing work, while others described it as an exhausting task outside of their expertise. We also heard that highly-visible evangelizer-maintainers can receive personal credit for all the group effort in a project. This Matthew effect [93] of accumulated status can generate tensions in the project, even if they do not intend for this to happen and actively work to elevate other contributors and maintainers.

4.8 Building and Maintaining Relationships: Meetups and Events

It is common for F/OSS projects to hold in-person meetups and events, often to get new and/or existing contributors together to accomplish work and build relationships. These events vary widely, from conferences to hackathons to happy hours. Our findings align with existing work that has also found these events play a critical role in developing trust and maintaining positive, lasting social relations [90, 91, 104]. Maintainers described in-person events as having an essential function and critical value in helping maintainers build good relationships, which helps them better understand each other in virtual environments [26]. Several longtime maintainers of major projects told us stories of their first F/OSS event, which they claimed inspired them to get even more involved in F/OSS in a way that years of online conversations had not. Past work has described the gendered labor women take on to organize events that often goes unacknowledged when technical contributions are valued above social and organizational work [92], which our participants also reflected upon.

Projects with many contributors, resources, and institutional connections often run their own major conferences, while smaller projects often meet up in more ad-hoc events. Smaller projects also rely on ecosystem-level conferences, in which those from various related projects (e.g. those written in the same programming language or that serve similar purposes) organize collective events. These ecosystem-level conferences often have dedicated periods for projects of various sizes to hold their own events. As projects grow the number of contributors, many move from holding satellite events before or after major F/OSS conferences to their own conferences.

In both our interviews and our observation of these events, we found that maintainers were often key organizers at events for smaller projects, although many larger projects with more capacity to fundraise hire dedicated event organizers. Projects that make connections to companies or universities often get in-kind donations of space and event organizing labor. Major projects with many users, contributors, and maintainers hold events that are more like trade conventions, with thousands of attendees and high competition for speaking slots. Some companies even specialize in hosting F/OSS conferences on behalf of projects. The companies then take a portion of the registration fees, which for larger projects can be over $1000 USD.

Interviewees told us it was often easier to get companies to fund events than anything else – testing infrastructure, time for labor, or the other costs that can accrue for a project. Many ecosystem-level F/OSS events are directly sponsored by companies that either rely extensively on F/OSS projects or are business arms of F/OSS projects. Maintainers discussed the labor associated with events as projects scale, as holding events for more and more people becomes increasingly difficult and costly. We also heard struggles maintainers face when the project has users, contributors, and maintainers across the world, which is another form of scaling that is often seen as a key metric of success. In-person events for a global community require more work, skills, and resources, including tasks like visas and fundraising for travel grants.

Some maintainers we interviewed shared how they spent significant amounts of their personal money on events they organized, particularly when funding promises fell through or when costs exceeded budgets. While expenses were sometimes reimbursed through donations, the actual labor of organizing events was less likely to be compensated or recognized. Some maintainers took issue

with restrictions from donors who would not fund stipends or even travel funds for those in their projects who organized an event, even if excess funds were available for expenses such as attendee travel, venues, and catering.

4.9 Funding, finances, and donations

Funding, finances, and donations were a major topic in our interviews, which has become the widely-discussed in broader public conversations in F/OSS. Our interviewees described funding as a way to compensate for labor already performed in maintaining F/OSS projects, as well as to pay contributors to perform tasks that are not done voluntarily. However, we found that fundraising and fund management itself can involve substantial amounts of unanticipated and specialized labor, including seeking funding, writing proposals and budgets, accounting, reimbursements, and managing relationships with funders. There is a strong parallel here to other non-profit sectors, including academic research, charities, and political organizations, where the work around funding can become a substantial fraction of the work performed in maintaining an organization. In academic research, however, scientists learn through their training that running a lab means maintaining a funding pipeline through networking, applications, and rejections [96]. Maintainers not trained in this system can be caught unprepared by the mismatch between their vision for the project and the reality of getting funding. Even for those who do not receive funding, the mere availability of potential funding can shift maintainers' and projects' conceptions of what F/OSS is and how their life and work fits into it. Funding can also push projects to develop more formal organizational structures, which in some cases can be at odds with their existing governance style and ethos.

4.9.1 Fundraising for maintenance: patronage and business models. We found two general approaches maintainers took to funding: patronage models and business models. In patronage, maintainers solicit donations or grants, while business models involve a range of strategies to sell services on top of F/OSS projects. While it may seem more obvious that building up a business involves a substantial amount of work and start-up costs, patronage models also can involve massive work in getting patrons and maintaining a good relationship with them. Companies, foundations, governments, and individuals all donate, but can have idiosyncratic processes around those donations. Managing the patron relationship can be a complex task as projects gain more patrons, particularly if patrons have contradictory expectations.

In our interviews, maintainers for projects that had funding to regularly hire multiple full-time employees expressed a common sentiment: they found themselves doing less and less work on the software project itself, and more and more work on seeking funding and managing the project. While we especially saw this with maintainers of large-scale projects in academic settings, we also encountered it in non-academic projects. Interviewees who actively sought grants and patronage told us that both grant agencies and other patrons often only fund novelty and new features, not the necessary upkeep, repair, security, and compatibility work. Maintainers struggle with producing the visions of novel innovation needed to get funding, which is a common theme in scientific cyberinfrastructure [9] and public works [39, 111].

This work around soliciting funding is not just about the raw amount of time or energy that maintainers spend trying to write proposals or find interested donors. There can be a heavy personal burden when maintainers become responsible for the livelihoods and careers of real people they have hired. One of our interviewees — an academic researcher whose grants fund employees to work on F/OSS — explained how getting funding can create an obligation to continue to get funding, to keep supporting people they hired. We particularly heard this in more academic-aligned F/OSS projects, where hiring graduate students or postdocs to work on F/OSS projects is common.

4.9.2 Money changes everything: the labor of spending. Once funding has been obtained and money is in some kind of account, the question of distribution and governance arises. Smaller projects grapple with learning how to navigate non-profit and for-profit laws around hiring, accounting, and taxes, requiring the project to bring in more kinds of expertise. As projects fundraise, maintainers can find themselves with obligations to expand their decision-making to include funders or those chosen by funders. This can be informal for smaller projects, but become more explicit as projects scale their fundraising. For example, the Linux Foundation has a Platinum membership level that costs $500,000 USD annually, and its corporate charter holds that about 80% of its Board of Directors are chosen by the Platinum members [50].

With projects that receive grants from more traditional foundations (whether private or public), the grant proposal already specifies how the funds will be spent. However, many projects receive more ad-hoc funding from donors who do not require extensive budgeted proposals, especially those that solicit funding through Patreon-style platforms like OpenCollective or GitHub Sponsors. As one maintainer explained, getting the funds was the easiest part:

> "...we created an OpenCollective, and a bunch of companies have contributed to it, but we didn't really address the issue of how to disburse the funds. [...] No real system for figuring out how to spend it. [...] Do we pay them [contributors] money for that one pull request that they did? [...] The problem doesn't solve itself just by the existence of money that's available for the project. There still has to be a mechanism and a policy for, like, distributing it among the people on the project."

The introduction of money into the project can bring the social relations of collaboration into conflict with labor and trade agreements. F/OSS projects often try to hire their long time volunteer contributors, no matter where they live. This means navigating through labor laws, varied immigration statuses, banking networks, or sanctions, which can be far more restrictive for some kinds of contributors than it is for others. Funding, then, transforms the structure of the organization, the possible formations of open source community, and what kinds of collaboration can sustain the maintenance of the project.

5 DISCUSSION: LABOR AND SCALE IN MAINTAINERSHIP

Our findings speak to two distinct but linked issues in F/OSS: labor and scale. Before we discuss more specific implications of our findings, we reflect about what we mean by the multi-faceted term "scale." Through our interviews about topics of labor, it became apparent that scale was clearly important, introduced by interviewees in response to a wide range of questions about many different aspects of their work and positions. Based on our interviews, "scale" can refer to: the number of people who use the software; the use of the software within large and/or prestigious organizations; the number of contributors or maintainers in the project; the number of bug reports, issues, and/or proposed changes made; the geographic distribution of users, contributors, and/or maintainers; the amount of rules and governance procedures; the number of communication channels used; the amount and/or rate of internal and external communication; the size, complexity, or features of the software code; and the interdependence of the code within a broader software ecosystem. Scale was also invoked as a more holistic feeling, particularly from those who described how they felt their project had grown too much too fast, making scale closer to a signifier of affect, as [122] also describe. Our findings advance work that interprets scale as a multidimentional quality beyond the number of users/participants [81], and methodologies that use participants' multiple understandings of scale as an analytic resource [86, 104].

Table 1. Summary of forms of work with examples of how they change as F/OSS projects scale.

	At smaller scales	At larger scales
The maintainer(s)	Solo or lead maintainer who makes all or most decisions, often by doing most of the work on their own.	Many maintainers with various divisions of labor, hierarchies, and organizational structures.
User support	An opportunity to retain and recruit new contributors. Work is ad-hoc.	An overwhelming flood. Work has established rules, teams, and triaging.
Managing software development	Governance is often implicit and led by a lead/solo maintainer, who accepts or rejects all proposed changes.	Governance is often explicitly discussed, with a variety of formal rules and structures for decision-making.
Code review and testing	Either no automated tests or lightweight tests managed by the lead/solo maintainer.	Widespread use of tests to review proposed changes and enforce rules. Managing testing is a dedicated role.
Ecosystem-level work	Projects may rely on other more-established projects and have to adapt to changes made "upstream."	Projects are embedded in an interdependent ecosystem, which must coordinate to ensure compatibility.
Evangelizing	A crucial task to get new users and contributors. Lead/solo maintainer must work to get speaking spots.	Maintainers are routinely invited to speak at conferences and prestigious organizations, some are celebrities.
Meetings and events	Smaller events focused on growing the user and contributor base, often organized by the lead/solo maintainer with little financial support.	Larger events that let contributors and maintainers coordinate and build relationships. Dedicated organizing roles with financial support.
Funding and finances	Small to non-existent. All work is uncompensated, but projects may receive donations for small expenses.	Routine and successful enough to hire contributors, maintainers, and accountants. Debates over how to raise and spend funds.

5.1 As projects scale, work not only increases, but fundamentally changes

As we summarize in Table 1, various kinds of work and positions of labor in F/OSS can become quite different at smaller and larger scales. Our findings extend prior work that investigates the different modes of scaling in technologically-mediated organizations and scientific cyberinfrastructure [3, 9, 75]. These similarities are potentially due to the fact that many F/OSS projects are also relied upon by decentralized communities (including science), where well-resourced user organizations and grant agencies contribute to their development and maintenance in a more ad-hoc fashion.

Carr and Lempert discuss how scale is not merely a matter of existing activities being amplified, but of work being fundamentally transformed by scale, because scale is deeply linked to power relations [122]. As F/OSS projects grow across many different understandings of scale, we showed how the kind of work involved in maintaining them also changes. For instance, in the findings we referred to an interviewee who shared how the growth of contributors to their F/OSS project necessitated the formalization and democratization of leadership positions when the "benevolent dictator" model was no longer sufficient, because it required all decisions to be approved by one person.

It is not simply that as projects grow, there is more work to be done, although this is also the case. New kinds of work are often needed and existing kinds of work become transformed. For

example, for a project with few users, providing user support to someone who raises an issue can be an exciting opportunity to grow the userbase. The sole maintainer likely does this work themselves, and has the capacity to attend to individual concerns. As F/OSS projects gain thousands or even millions of users, maintainers often must implement distributed approaches, like directing questions to Q&A sites or forming teams who solely triage the issue queue. This is also the case with continuous integration (CI), where when projects grow their codebase, interdependence, and contributors, more tests must be run, which exceed the free allowances of commercial CI offerings. This may initially seem to be a purely "technical" challenge, but raises questions about fundraising and organizational roles.

The CI example also illustrates the deeply socio-technical nature of work, which has long been an established concept in CSCW and organization studies [7, 98, 108, 133]. Our findings extend the literature on how maintenance and repair practices are bound up in social relationships [37, 66, 71, 72, 99, 110, 120]. One way of understanding this implication is that there is no "purely technical" work in F/OSS that only requires software engineering expertise, as all forms of work have interpersonal and organizational dimensions, even if those are often implicit.

5.2 Scalar labor: What is needed to grow in many directions?

Many F/OSS projects have a small user base and do not grow beyond a single maintainer-contributor [42], but as the prior section discussed, those that do become widely relied upon as infrastructure must be maintained with increasingly more work and different kinds of work. When this occurs, the maintainer(s) must constantly ensure there are enough people available and willing to work on what the project needs. Those people must also have the skills, resources, institutional knowledge, and organizational forms necessary for them to do that work well. We introduce the term "scalar labor" to describe these kinds of work that seek to ensure the project has capacity to meet its many growing needs, across the many dimensions in which the project may scale. We use "scalar" primarily as an adjective form of scale, but its mathematical meaning as magnitude without a specified direction can be an apt metaphor in F/OSS, particularly for projects that achieve what one interviewee called "catastrophic success."

The concept of scalar labor overlaps with Ribes's focus on "scalar devices" [104] in his studies of scientific cyberinfrastructure, which are the tools and practices that people in organizations use to understand and manage the size, scope, and spread of the organization. These include surveys, all-hands meetings, analyses of logs and digital traces, and other "little technologies of community." Like other works on scaling, Ribes discusses the many heterogeneous dimensions that people tend to compress into the single term. While Ribes's article is more methodologically focused on how ethnographers can study such organizations through scalar devices, we also found many of the same empirical findings in our ethnography of a similar kind of process, in a different set of social worlds.

In both F/OSS and scientific cyberinfrastructure, there can be a prevalent assumption that scaling-up is an inherent good, which is rewarded by funders who support projects that demonstrate successful scaling. Ribes also discusses how it can be far more difficult to manage projects as they seek to scale-up, particularly when scaling into becoming infrastructure for an entire academic discipline. Ribes' "scalar device" is what sociologists call a sensitizing concept [11] that draws our attention more to knowing what scale an organization is at now, what it could be at in the future, and how those within it know the organization. Scalar labor, by contrast, draws our attention more to how this work transforms with a changing organizational, economic, and institutional context. Labor also calls attention to who does this work, who is recognized for doing this work, what it costs them, and how they are compensated or made whole for doing it.

Scalar labor is also related to Strauss's concept of "articulation work" [118], which Gerson summarises as "making sure all the various resources needed to accomplish something are in place and functioning where and when they're needed" [58]. Bietz et al.'s study of scientific cyberinfrastructure [9] introduces a related extension of articulation work in "synergizing," which is the work of creating and maintaining a common field where quite different kinds of people, organizations, and systems can do articulation work. Synergizing calls our attention to how this work is impacted by the heterogeneity of interdependent people, organizations, and systems that must be coordinated, which was also certainly a theme in our findings. By emphasizing the labor dimension of F/OSS, scalar labor covers a similar range of activities as synergizing, but draws our attention to how this work is specifically impacted by the heterogeneity of different dimensions of growth, of which interdependence is but one mode of scaling.

Like articulation work and synergizing, scalar labor is complex and interdependent. A prime example of these phenomena is raising funds to host an event to evangelize to new users, who would then be mentored into contributors, who would then respond to bug reports and fix identified issues, and may even become mentored into maintainers themselves. In this example, a traditional software engineering firm that made money from selling licenses or services would simply hire someone directly to respond to bug reports and fix such issues. Some F/OSS projects with significant fundraising capacity do exactly this, which can relieve major burdens. However, most projects we encountered could only recruit volunteers, because they cannot charge for the free software and also struggle to achieve the status that would help them fundraise. In either case, the concept of scalar labor draws our attention to how growth is not sought for its own sake, but rather as a strategy to build capacity so that important maintenance work can be done. Yet because this growth itself can bring more and different kinds of work, even more growth may be needed to do that work.

Also like articulation work and synergizing, scalar labor is a useful concept for studying F/OSS (and other organizations that produce and maintain infrastructure) because it includes interpersonal, organizational, and financial skills that are often far outside of the scope of a traditional engineer's duties — even though it is done to improve the project's capacity to do traditional engineering work. Yet this work also often requires project-specific knowledge and the trust of contributors, which makes it difficult to delegate. This work can becomes what maintainers call "governance" – a longstanding topic in F/OSS research (e.g. [87]. Yet despite such a focus on governance, governance work is rarely analyzed as a form of labor, perhaps because it is seen to be more about organizational forms or decision-making. Kelty [76] is notable exception as he details how free software projects often make and enact governance decisions through software engineering, and they recognize those engineering decisions as constituting social values. The concept of scalar labor calls our attention to governance as a form of labor, which can be as much of an an exhausting, uncompensated, and invisible burden as it is the exercise of power.

5.3 Scalar debt and the consequences of 'catastrophic success'

The concept of scalar labor leads to a related issue: many projects that become widely relied upon accumulate what we call "scalar debt," a term we introduce based on the concept of "technical debt" [33]. Technical debt refers to engineering decisions that initially help a project advance or expand quickly, but at a cost that must be 'paid back' later with even more work than was initially saved. Projects that grow rapidly and achieve what one interviewee called "catastrophic success" struggle in that they have not done enough scalar labor; that is, their growth in users has not led to a growth in the project's capacity to maintain itself at this new scale. Paying down this scalar debt then comes at an immense cost to the maintainers, which in contemporary F/OSS parlance, is often referred to as finding a working "sustainability model" — a way to recruit new volunteers or raise

funds to hire contributors and maintainers to do work that is currently backlogged. This focus on the present is different from how "sustainability" is discussed in scientific cyberinfrastructure, where it usually refers to questions about whether the infrastructure will persist in the long-term, often future decades [105, 106].

Scalar debt also is related to how F/OSS projects tend to develop management and governance structures on an ad-hoc basis over time, rather than preemptively plan these out before they are needed. This ad-hoc or "spontaenous" [35] governance model is common in F/OSS, as well as other peer production platforms [17, 124]. Researchers and practitioners in F/OSS have identified this a key strength [21, 130] — with some comparisons to the now-dominant Agile software development methodology [52, 129]. This ad-hocness in F/OSS has also been described as a product of shared ideological and cultural commitments, whose members may object to formalization and instead value autonomy and more distributed governance models [27, 35].

However, this ad-hocness is also related to the resources and labor available to projects in the mostly-voluntary peer-production model that the F/OSS projects we studied began with. In our interviews, maintainers told us their concerns around how such formalization and bureaucratization can take substantial time, energy, social capital, and risk, meaning that there are very good reasons why a project may not want create such structures unless it is absolutely apparently necessary. However, one key case of scalar debt we encountered in multiple interviews was around Codes of Conduct and other moderation mechanisms, which have become particularly central to discussions of diversity and inclusion in F/OSS [38].

It can be difficult to know when a project is taking on scalar debt, although once this becomes apparent, maintainers described how they were living in a constant state of incipient crisis, overwork, or burnout just to keep the projects from falling too far backwards. Ironically, when projects are in this state of "putting out fires," more work must be done to get and manage resources necessary to help put out those fires, whether this involves recruiting and mentoring new volunteer contributors or raising funds to hire employees. The success of these sustainability approaches are not guaranteed: volunteers can leave, patrons can withdraw support, grants can be rejected, and business models can fail to be profitable. For maintainers in the overwhelming 'putting out fires' state, it can be a difficult choice as to whether they spend their scarce time and energy on an unproven strategy to leverage more resources for the project, versus spending that time putting out the fires currently flaring up. As [122] and [104] also discuss in non-F/OSS contexts, projects have to continuously recalibrate what scale they are currently at, which also can be an uncertain and labor-intensive task.

5.4 Scaling into becoming critical infrastructure for well-resourced organizations

This issue of scalar debt leads to a related issue that becomes apparent when a project scales into becoming relied upon as critical infrastructure by well-resourced organizations, from for-profit companies to universities to governments. As we previously reviewed, entire sectors of the global economy are now reliant on F/OSS projects. We use the term "labor" intentionally in this paper, which calls attention to how this work is part of the economy, contributing to the production and distribution of goods and services. In contrast, earlier work on F/OSS often emphasized the voluntary, altruistic, and alternative nature of this work, framing these projects more as communities. While the rising commercialization of F/OSS as a movement is a long and well-studied historical trend over the past two decades [10, 47, 77], we find that contemporary projects which begin as more voluntary, peer-production efforts can similarly transform as they scale. Early on, many smaller and more voluntary F/OSS projects seek to be relied upon, especially by large, well-known tech companies and universities, who can directly or indirectly help support the project. This can

bring not only users (who may become contributors), but also prestige, connections, a pittance of donations — that is, cultural, social, and financial capital [13].

Yet several maintainers we interviewed described it as "blessing and a curse" to be relied upon by organizations that build their products on their projects. The companies benefited from their labor, but often did not offer resources or labor in return. We often heard how being relied upon in such a manner adds more work that is also more difficult; that software engineers at elite user organizations can be especially demanding and entitled. Maintainers must respond to these elite users in ways addressed in studies of emotional labor, that is, in managing the emotions of others [65]. Many such user organizations are "free riders" who do not contribute back. Even when some corporations contribute back to F/OSS, they can do so in ways that place additional demands on maintainers, by demanding additional code review or asking F/OSS contributors to managing relationships with patrons. These are all ways in which becoming critical infrastructure for well-resourced organizations can increase and transform the work of maintainers, even as it brings users, resources, and prestige.

Another issue that arises is that becoming relied upon as infrastructure by others can make maintainers feel morally responsible for whatever products are built using their projects. Some described becoming disillusioned or burned-out specifically because they began their project imagining it would be used by those who could not afford commercial alternatives, but found it was more used by companies to lower their own costs. Some even expressed that they had to wrestle with how their projects were used as part of products the maintainers believed were unethical or harmful. These frustrations are compounded by maintainers' frustrations that the wealth derived by companies relying upon their technologies does not get shared with maintainers/contributors, evidencing an inequitable form of extraction. The activities around mentally working through such issues can also be seen as a form of both invisible work and scalar labor.

5.5 Scaling and the dynamics of hypervisibility

Scale impacts maintainers' personal identities and relationships to broader publics. While much prior literature on maintenance and infrastructure in other sectors (e.g. power plants, transportation, commercial software) discusses maintenance as relatively less-visible and less-recognized work [39, 111, 117, 120], the centrality of maintainers as leaders of F/OSS projects leads to a different set of issues. Much of F/OSS work is done in public view because of the open nature of F/OSS work and especially the dominance of all-inclusive public code collaboration platforms like GitHub. As discussed in the sections on user support and proposed changes, maintainers can receive a deluge of requests from users and contributors, all of which are visible on the public web. Under public scrutiny, maintainers engage in the communicative labor of tracking management, emotional labor of user and contributor response, and, in sum, the production of the optics of a successful project.

Maintainers of projects that have achieved massive success and scale — that are widely relied-upon and/or have a large contributor base — can achieve a kind of "microcelebrity" [89] status, a term originally from studies of social media. Eghbal compares F/OSS maintainers to content creators on Youtube or Instagram, particularly those who earn a quasi-independent living through patronage [42]. We found that some maintainers grow into microcelebrities, fueled by the dynamics of social media and technology standardization. Such maintainers have hundreds of thousands of followers on social media sites like Twitter, write widely-read blog posts on the state of F/OSS, and are flown to speak at major conferences and companies. Such maintainers can play a major role in conflict resolution and governance issues on public platforms such as mailing lists, particularly when the governance model is more ad-hoc. These influential leaders can become substitutes for the reorganization of project decision making and conflict resolution – a case of scalar debt.

Evangelizing can reinforce this trend towards hypervisible microcelebrity maintainers, such as if an already famous maintainer is invited to give talks at F/OSS conferences to thousands of people, or flown out to give talks at companies and universities, which makes them even more famous. Funding, patronage and business models benefit from more famous figureheads, as well as often require that a single individual is the designated Principal Investigator on the grant or CEO of the business arm. Some microcelebrity maintainers told us they have to actively work against these dynamics, such as by sending others in their place when asked to speak at conferences.

Yet there still are forms of invisible work for the hypervisible. Such maintainers routinely receive torrents of unsolicited e-mails and private messages, from lavish praise to harassment. Much work also takes place outside of public code platforms, like writing grants or conflict resolution. Precisely because of their microcelebrity, these maintainers can be called in to adjudicate disputes behind the scenes. These findings suggest that maintenance labor is not always invisible; it can be hypervisible and highly valued. Given the dominant framing of maintenance and infrastructure as invisible work, [60, 116, 128] we urge future research into this intersection of issues.

6 CONCLUSION

The focus of this paper has been on the intersection of labor and scale in the context of maintaining F/OSS projects, but our findings contribute to understanding challenges faced by people engaging in a variety of types of collaborative work to build common information resources while simultaneously developing organizations and governance structures. In our interviews, maintainers described being burned out by changes in what was expected of them fundamentally changed as projects scaled. These interviews were rich with insights into the deep and varied commitments of F/OSS maintainers, but also the emotional toll doing F/OSS work can take. Our findings have a wide import for discussions of governance, leadership, and sustainability, in socio-technical systems, including crowdsourcing, citizen science, scientific cyberinfrastructure, and crisis informatics. Particularly, our focus on labor - and people's reactions to changes in their labor - can help build awareness of how infrastructure sustainability is tied to the long-term well-being of maintainers as individuals and in their communities.

6.1 Limitations

Although we attempted to recruit a diverse group of participants for our interviews — with particular attention to the type/size of F/OSS project they worked on, employment, and geography — our findings are limited by the number of interviews we conducted and our recruitment methods. We have mostly studied projects that have been relied upon by others as infrastructure and began as volunteer projects, so our findings do not speak to overwhelming majority F/OSS projects that are developed and used by a single person but released publicly, as well as to entirely corporate-developed F/OSS projects. We have also sought to capture a kind of longitudinal view by focusing on maintainers, some of whom have long histories of involvement. A more traditional longitudinal study would capture these issues of scale with even more depth. Like in all interview-based studies, memories may be less accurate, so this study could be complemented with more detailed contemporaneous methods of capturing the work that maintainers do day-to-day, from participant-observation to diary studies to analyses of trace data.

We also acknowledge how we are implicated in the same kinds of systems of F/OSS sustainability as our participants. All authors have direct participant experience in F/OSS projects as contributors or maintainers, which gives us a sensitivity to these topics, but also means that we can lack some analytical distance that some strands of social science value. In particular, the fact that we were funded to study these issues by non-profit foundations that are also direct funders of F/OSS projects

— which was public knowledge that we disclosed prior to our interviews — may impact the kinds of responses we received.

6.2 Recommendations and future work

Contributors and maintainers might better manage difficulties posed by scale if they regularly have conversations about what responsibilities entail, how much time and effort that work takes, and how the distribution of workloads and resources should change when the project changes. Maintainers may benefit from explicitly acknowledging when scalar debt is being taken on, as is sometimes commonly acknowledged when technical debt is being taken on. Focusing on these questions of scalar labor brings to light how scale is not always a universally good thing — even though there are broad pressures on projects that equate scale with success, as [104] also discusses in science. The benefits of scaling and success many also not be equitably distributed, as we discussed around the less visible and more gendered labor of event organizing, versus the dynamics that lead to microcelebrity maintainers. Finally, because efforts to build capacity and reduce the burdens of maintenance work itself can compound the amount of work to be done, funders and donors can be mindful of the opportunity costs that projects spend in soliciting resources. This can involve more lightweight funding mechanisms that require less up-front work on the part of maintainers and project leaders.

Many areas in this paper might be expanded in future work. Specifically, we are interested in unpacking the effects of corporate reliance on F/OSS projects on maintainers' working and emotional lives. Although we brought in value misalignment as one way to interpret maintainers' reactions when corporations took but didn't give back to F/OSS, we believe more work can be done in this area to understand the political economy of value misalignment and the effects of corporate reliance on maintainers' mental health and well-being. This might involve conducting additional interviews that focus on projects' growth trajectories or focusing on projects that experienced the 'catastrophic success' gestured to in the discussion. Further exploring these areas might contribute valuable and actionable insights to improve F/OSS sustainability.

7 ACKNOWLEDGMENTS

The authors would like to thank Alexandra Paxton, Nelle Varoquaux, Chris Holdgraf for their ongoing feedback, as well as Linwei Lu, Julio Gonzalez, and the CSCW reviewers for their insights. We are thankful to the cohort, advisors, and program managers of the Ford/Sloan Critical Digital Infrastructures Initiative in helping us plan this research. We appreciate the time our anonymous interviewees spent talking with us and reviewing various drafts of this work. We are thankful to Stacey Dorton for administrative support. This work has been financially supported by the Ford and Sloan Foundation through the Critical Digital Infrastructures Initiative (grant G-2018-11354), the National Science Foundation (grant DDRIG #1947213), as well as the Gordon & Betty Moore Foundation (grant GBMF3834) and Alfred P. Sloan Foundation (grant 2013-10-27) through the Moore-Sloan Data Science Environments grant to UC-Berkeley.

REFERENCES

[1] Morgan G. Ames, Daniela K. Rosner, and Ingrid Erickson. 2015. Worship, faith, and evangelism: Religion as an ideological lens for engineering worlds. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, New York, 69–81. https://doi.org/10.1145/2675133.2675282

[2] Brian Anderson. 2017. How Rust is tested. https://brson.github.io/2017/07/10/how-rust-is-tested

[3] Karen S Baker, David Ribes, Florence Millerand, and Geoffrey Bowker. 2005. Interoperability strategies for scientific cyberinfrastructure: Research and practice. In *Proceedings of the American Society for Information Science and Technology* (2005). https://doi.org/10.1002/meet.14504201237

[4] Flore Barcellini, Françoise Détienne, and Jean-Marie Burkhardt. 2014. A situated approach of roles and participation in open source software communities. *Human–Computer Interaction* 29, 3 (2014), 205–255. https://doi.org/10.1080/07370024.2013.812409
[5] Ann Barcomb. 2016. Episodic volunteering in open source communities. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. 1–3. https://doi.org/10.1145/2915970.2915972
[6] Yochai Benkler. 2007. *The Wealth of Networks: How Social Production Transforms Markets and Freedom.* Yale University Press.
[7] Richard Bentley, John A. Hughes, David Randall, Tom Rodden, Peter Sawyer, Dan Shapiro, and Ian Sommerville. 1992. Ethnographically-informed systems design for air traffic control. In *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work*. 123–129. https://doi.org/10.1145/143457.143470
[8] Magnus Bergquist and Jan Ljungberg. 2001. The power of gifts: Organizing social relationships in open source communities. *Information Systems Journal* 11, 4 (2001), 305–320. https://doi.org/10.1046/j.1365-2575.2001.00111.x
[9] Matthew J Bietz, Eric PS Baumer, and Charlotte P Lee. 2010. Synergizing in cyberinfrastructure development. *Computer Supported Cooperative Work (CSCW)* 19, 3-4 (2010), 245–281. https://doi.org/10.1007/s10606-010-9114-y
[10] Benjamin J. Birkinbine. 2015. Conflict in the Commons: Towards a Political Economy of Corporate Involvement in Free and Open Source Software. *The Political Economy of Communication* 2, 2 (2015). http://www.polecom.org/index.php/polecom/article/view/35 Number: 2.
[11] Herbert Blumer. 1954. What is wrong with social theory? *American Sociological Review* 19, 1 (1954), 3–10.
[12] Herbert Blumer. 1969. *Symbolic Interactionism: Perspective and Method.* University of California Press, Berkeley.
[13] Pierre Bourdieu. 1973. Cultural reproduction and social reproduction. In *Knowledge, Education, and Cultural Change*, Richard Brown (Ed.). London: Tavistock.
[14] Geoffrey C. Bowker and Susan Leigh Star. 2000. *Sorting Things Out: Classification and Its Consequences.* MIT Press.
[15] Daren C Brabham. 2013. *Crowdsourcing.* The MIT Press, Cambridge, MA.
[16] Dale A Bradley. 2006. The divergent anarcho-utopian discourses of the open source software movement. *Canadian Journal of Communication* 30, 4 (2006).
[17] A. Bruckman and A. Forte. 2008. Scaling consensus: Increasing decentralization in Wikipedia governance. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS)* (2008). 157.
[18] Julia Bullard. 2016. Motivating invisible contributions: Framing volunteer classification design in a fanfiction repository. In *Proceedings of the 19th International Conference on Supporting Group Work* (Sanibel Island, Florida, USA, 2016-11-13) *(GROUP '16)*. ACM, 181–193. https://doi.org/10.1145/2957276.2957295
[19] Brett Cannon. 2017. The give and take of open source. Talk at JupyterCon 2017. O'Reilly Media. https://www.oreilly.com/radar/the-give-and-take-of-open-source/
[20] Andrea Capiluppi and Martin Michlmayr. 2007. From the cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects. In *Open Source Development, Adoption and Innovation.* Springer US, 31–44. https://doi.org/10.1007/978-0-387-72486-7_3
[21] Eugenio Capra, Chiara Francalanci, and Francesco Merlo. 2008. An empirical study on the relationship between software design quality, development effort and governance in open source projects. *IEEE Transactions on Software Engineering* 34, 6 (2008), 765–782.
[22] Adele E Clarke. 2003. Situational analyses: Grounded theory mapping after the postmodern turn. *Symbolic Interaction* 26, 4 (2003), 553–576.
[23] Adele E Clarke and Susan Leigh Star. 2008. The social worlds framework: A theory/methods package. In *The Handbook of Science and Technology Studies.* MIT Press, Cambridge, MA, 113–137.
[24] Gabriella Coleman. 2004. The political agnosticism of free and open source software and the inadvertent politics of contrast. *Anthropological Quarterly* 77, 3 (2004), 507–519.
[25] Gabriella Coleman. 2009. Code is speech: Legal tinkering, expertise, and protest among free and open source software developers. *Cultural Anthropology* 24, 3 (2009), 420–454.
[26] Gabriella Coleman. 2010. The hacker conference: A ritual condensation and celebration of a lifeworld. *Anthropological Quarterly* (2010), 47–72.
[27] Gabriella Coleman. 2012. *Coding Freedom: The Ethics and Aesthetics of Hacking.* Princeton University Press, Princeton.
[28] The Kernel Development Community. 2018. How the development process works. The Linux Kernel documentation. https://www.kernel.org/doc/html/v4.15/process/2.Process.html
[29] Kevin Crowston. 2011. Lessons from volunteering and free/libre open source software development for the future of work. In *Researching the Future in Information Systems* (Berlin, Heidelberg, 2011) *(IFIP Advances in Information and Communication Technology)*, Mike Chiasson, Ola Henfridsson, Helena Karsten, and Janice I. DeGross (Eds.). Springer, 215–229. https://doi.org/10.1007/978-3-642-21364-9_14
[30] Kevin Crowston, Robert Heckman, Hala Annabi, and Chengetai Masango. 2005. A structurational perspective on leadership in Free/Libre Open Source Software teams. *Proceedings of the First International Conference on Open Source*

Systems (2005), 7.
[31] Kevin Crowston, Qing Li, Kangning Wei, U. Yeliz Eseryel, and James Howison. 2007. Self-organization of teams for free/libre open source software development. *Information and Software Technology* 49, 6 (2007), 564–575. https://doi.org/10.1016/j.infsof.2007.02.004
[32] 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. *ACM Computing Surveys (CSUR)* 44, 2 (March 2012), 35. https://doi.org/10.1145/2089125.2089127
[33] Ward Cunningham. 1992. The WyCash portfolio management system. In *Proceedings of the Object-oriented Programming Systems, Languages, and Applications (Addendum)* (Vancouver, British Columbia, Canada, 1992-12-01) *(OOPSLA '92)*. Association for Computing Machinery, 29–30. https://doi.org/10.1145/157709.157715
[34] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. ACM, New York, 1277–1286.
[35] Paul B. de Laat. 2007. Governance of open source software: state of the art. *Journal of Management & Governance* 11, 2 (2007), 165–177. https://doi.org/10.1007/s10997-007-9022-9
[36] Luiz Felipe Dias, Igor Steinmacher, Gustavo Pinto, Daniel Alencar da Costa, and Marco Gerosa. 2016. How does the shift to github impact project collaboration?. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 473–477.
[37] Fernando Domínguez Rubio. 2020. *Ecologies of the Modern Imagination at the Art Museum*. University of Chicago Press, Chicago.
[38] Christina Dunbar-Hester. 2019. *Hacking Diversity: The Politics of Inclusion in Open Technology Cultures*. Vol. 21. Princeton University Press.
[39] David Edgerton. 2011. *Shock of the Old: Technology and Global History Since 1900*. Oxford University Press, Oxford.
[40] Paul N Edwards, Steven J Jackson, Geoffrey C Bowker, and Cory P Knobel. 2007. Understanding infrastructure: Dynamics, tensions, and design. *Report of NSF Workshop on "History & Theory of Infrastructure: Lessons for New Scientific Cyberinfrastructures"* (2007). https://deepblue.lib.umich.edu/bitstream/handle/2027.42/49353/UnderstandingInfrastructure2007.pdf
[41] Nadia Eghbal. 2016. *Roads and bridges: The unseen labor behind our digital infrastructure*. Ford Foundation.
[42] Nadia Eghbal. 2020. *Working in Public: The Making and Maintenance of Open Source Software*. Stripe Press.
[43] Hamid R Ekbia and Bonnie A Nardi. 2017. *Heteromation, and Other Stories of Computing and Capitalism*. MIT Press.
[44] Nathan Ensmenger. 2008. Fixing things that can never be broken: Software maintenance as heterogeneous engineering. In *Proceedings of the SHOT Conference*.
[45] Joseph Feller, Patrick Finnegan, Brian Fitzgerald, and Jeremy Hayes. 2008. From Peer Production to Productization: A Study of Socially Enabled Business Exchanges in Open Source Service Networks. *Information Systems Research* 19, 4 (2008), 475–493. https://doi.org/10.1287/isre.1080.0207
[46] Anna Filippova and Hichang Cho. 2015. Mudslinging and Manners: Unpacking Conflict in Free and Open Source Software. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing* (Vancouver, BC, Canada, 2015-02-28) *(CSCW '15)*. ACM, 1393–1403. https://doi.org/10.1145/2675133.2675254
[47] Brian Fitzgerald. 2006. The Transformation of Open Source Software. *MIS Quarterly* 30, 3 (2006), 587–598. https://doi.org/10.2307/25148740
[48] Lee Fleming and David M Waguespack. 2007. Brokerage, boundary spanning, and leadership in open innovation communities. *Organization Science* 18, 2 (2007), 165–180.
[49] Karl Fogel. 2005. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media.
[50] Linux Foundation. [n.d.]. The Bylaws of the Linux Foundation. https://www.linuxfoundation.org/en/bylaws/
[51] Sarah E. Fox, Kiley Sobel, and Daniela K. Rosner. 2019. Managerial Visions: Stories of upgrading and maintaining the public restroom with IoT. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–15.
[52] Erich Gamma. 2005. Agile, open source, distributed, and on-time: Inside the eclipse development process. In *International Conference on Software Engineering: Proceedings of the 27th International Conference on Software Engineering*, Vol. 15. 4–4.
[53] Juan Mateos Garcia, W Edward Steinmueller, et al. 2003. *The open source way of working: A new paradigm for the division of labour in software development?* SPRU.
[54] R. Stuart Geiger. 2011. The Lives of Bots. In *Wikipedia: A Critical Point of View*, G. Lovink and N. Tkacz (Eds.). Institute of Network Cultures, 78–93. http://www.stuartgeiger.com/lives-of-bots-wikipedia-cpov.pdf
[55] R. Stuart Geiger and David Ribes. 2011. Trace ethnography: Following coordination through documentary practices. In *2011 44th Hawaii International Conference on System Sciences*. IEEE, 1–10.
[56] Matt Germonprez, Julie E. Kendall, Kenneth E. Kendall, Lars Mathiassen, Brett Young, and Brian Warner. 2016. A Theory of Responsive Design: A Field Study of Corporate Engagement with Open Source Communities. *Information*

Systems Research 28, 1 (2016), 64–83. https://doi.org/10.1287/isre.2016.0662 Publisher: INFORMS.

[57] Matt Germonprez, Georg J.P. Link, Kevin Lumbard, and Sean Goggins. 2018. Eight Observations and 24 Research Questions About Open Source Projects: Illuminating New Realities. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 57 (2018), 22 pages. https://doi.org/10.1145/3274326

[58] Elihu M Gerson. 2008. Reach, bracket, and the limits of rationalized coordination: Some challenges for CSCW. In *Resources, Co-Evolution and Artifacts*. Springer, 193–220.

[59] Paola Giuri, Francesco Rullani, and Salvatore Torrisi. 2008. Explaining leadership in virtual teams: The case of open source software. *Information Economics and Policy* 20, 4 (2008), 305–315.

[60] Stephen Graham and Nigel Thrift. 2007. Out of order: Understanding repair and maintenance. *Theory, Culture & Society* 24, 3 (2007), 1–25.

[61] Kaj Grønbæk, Morten Kyng, and Preben Mogensen. 1992. CSCW challenges in large-scale technical projects—a case study. In *Proceedings of the 1992 ACM Conference on Computer-supported Cooperative Work*. 338–345.

[62] Scott Hanselman. 2015. *Bring Kindness back to Open Source.* https://www.hanselman.com/blog/bring-kindness-back-to-open-source

[63] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)*. Association for Computing Machinery, New York, NY, USA, 426–437. https://doi.org/10.1145/2970276.2970358

[64] Eric von Hippel. 2001. Innovation by User Communities: Learning from Open-Source Software. *MIT Sloan Management Review* 42, 4 (2001), 82–82. https://go.gale.com/ps/i.do?p=AONE&sw=w&issn=15329194&v=2.1&it=r&id=GALE%7CA77578225&sid=googleScholar&linkaccess=abs Sloan Management Review.

[65] Arlie Russell Hochschild. 1983. *The Managed Heart: Commercialization Of Human Feeling*. University of California Press, Oakland, California.

[66] Lara Houston, Steven J. Jackson, Daniela K. Rosner, Syed Ishtiaque Ahmed, Meg Young, and Laewoo Kang. 2016. Values in Repair. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA, 2016-05-07) *(CHI '16)*. ACM, 1403–1414. https://doi.org/10.1145/2858036.2858470

[67] Dorothy Howard and R. Stuart Geiger. 2019. Ethnography, Genealogy, and Political Economy in the Post-Market Era of Free & Open-Source Software. In *Proceedings of CSCW '19 Extended Abstracts*.

[68] Dorothy Howard and Lilly Irani. 2019. Ways of Knowing When Research Subjects Care. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–16.

[69] James Howison. 2015. Sustaining scientific infrastructures: transitioning from grants to peer production. In *iConference 2015* (2015-03-15). https://www.ideals.illinois.edu/handle/2142/73439 Accepted: 2015-03-23T21:58:14Z Publisher: iSchools.

[70] Lilly C Irani and M Six Silberman. 2013. Turkopticon: Interrupting worker invisibility in amazon mechanical turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 611–620.

[71] Lilly C. Irani and M. Six Silberman. 2016. Stories we tell about labor: Turkopticon and the trouble with "design". In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA, 2016-05-07) *(CHI '16)*. ACM, 4573–4586. https://doi.org/10.1145/2858036.2858592

[72] Steven J. Jackson, Syed Ishtiaque Ahmed, and Md. Rashidujjaman Rifat. 2014. Learning, innovation, and sustainability among mobile phone repairers in Dhaka, Bangladesh. In *Proceedings of the 2014 conference on Designing interactive systems* (Vancouver, BC, Canada, 2014-06-21) *(DIS '14)*. Association for Computing Machinery, 905–914. https://doi.org/10.1145/2598510.2598576

[73] Steven J Jackson, Alex Pompe, and Gabriel Krieshok. 2012. Repair worlds: maintenance, repair, and ICT for development in rural Namibia. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. 107–116.

[74] C. Jensen and W. Scacchi. 2005. Collaboration, Leadership, Control, and Conflict Negotiation and the Netbeans.org Open Source Software Development Community. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences* (2005-01). 196b–196b. https://doi.org/10.1109/HICSS.2005.147 ISSN: 1530-1605.

[75] Helena Karasti, Karen S. Baker, and Florence Millerand. 2010. Infrastructure time: Long-term matters in collaborative development. *Computer Supported Cooperative Work (CSCW)* 19, 3 (2010), 377–415. https://doi.org/10.1007/s10606-010-9113-z

[76] Christopher M Kelty. 2008. *Two Bits: The Cultural Significance of Free Software.* Duke University Press.

[77] Christopher M Kelty. 2013. There is no free software. *The Journal of Peer Production* (2013). Issue 3. http://peerproduction.net/issues/issue-3-free-software-epistemics/debate/there-is-no-free-software/

[78] Mathias Klang. 2005. Free software and open source: The freedom debate and its consequences. *First Monday* 10, 3 (2005).

[79] Nolan Lawson. 2017. What it feels like to be an open-source maintainer. Read the Tea Leaves. https://nolanlawson.com/2017/03/05/what-it-feels-like-to-be-an-open-source-maintainer/

[80] 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* (Banff, Alberta, Canada) *(CSCW '06)*. ACM, New York, NY, USA, 483–492. https://doi.org/10.1145/1180875.1180950

[81] Charlotte P Lee and Drew Paine. 2015. From The matrix to a model of coordinated action (MoCA) A conceptual framework of and for CSCW. In *Proceedings of the 18th ACM Conference on Computer-supported Cooperative Work & Social Computing*. 179–194.

[82] Yan Li, Chuan-Hoo Tan, and Hock-Hai Teo. 2012. Leadership characteristics and developers' motivation in open source software development. *Information & Management* 49, 5 (2012), 257–267.

[83] Yu-Wei Lin, Jo Bates, and Paula Goodale. 2016. Co-observing the weather, co-predicting the climate: Human factors in building infrastructures for crowdsourced data. *Science and Technology Studies* 29, 3 (2016), 10–27. http://dspace.stir.ac.uk/handle/1893/26101 Accepted: 2017-11-28T23:28:19Z Publisher: Finnish Society for STS.

[84] Arwid Lund. 2017. *Wikipedia, Work and Capitalism*. Springer, London.

[85] Jennifer Helene Maher. 2015. *Software Evangelism and the Rhetoric of Morality: Coding Justice in a Digital Democracy*. Routledge, London.

[86] George E. Marcus. 1995. Ethnography in/of the World System: The Emergence of Multi-Sited Ethnography. *Annual Review of Anthropology* 24, 1 (1995), 95–117. https://doi.org/10.1146/annurev.an.24.100195.000523

[87] M Lynne Markus. 2007. The governance of free/open source software projects: monolithic, multidimensional, or configurational? *Journal of Management & Governance* 11, 2 (2007), 151–163.

[88] Steve Marquess. 2014. *Of Money, Responsibility, and Pride*. http://veridicalsystems.com/blog/of-money-responsibility-and-pride/ Library Catalog: veridicalsystems.com.

[89] Alice Marwick and Danah Boyd. 2011. To see and be seen: Celebrity practice on Twitter. *Convergence* 17, 2 (2011), 139–158.

[90] Ashwin Mathew and Coye Cheshire. 2017. Risky Business: Social Trust and Community in the Practice of Cybersecurity for Internet Infrastructure. IEEE. https://doi.org/10.24251/HICSS.2017.283

[91] Ashwin J. Mathew. 2016. The myth of the decentralised internet. 5, 3 (2016). https://policyreview.info/articles/analysis/myth-decentralised-internet

[92] Amanda Menking and Ingrid Erickson. 2015. The heart work of Wikipedia: Gendered, emotional labor in the world's largest online encyclopedia. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, ACM*. 207–210.

[93] Robert K Merton. 1968. The Matthew effect in science: The reward and communication systems of science are considered. *Science* 159, 3810 (1968), 56–63.

[94] Audris Mockus, Roy T. Fielding, and James Herbsleb. 2000. A case study of open source software development: the Apache server. In *Proceedings of the 22nd International Conference on Software Engineering* (Limerick, Ireland, 2000-06-01) *(ICSE '00)*. Association for Computing Machinery, 263–272. https://doi.org/10.1145/337180.337209

[95] Lauren Morse, Janice M & Clark. 2019. The nuances of grounded theory sampling and the pivotal role of theoretical sampling. *The SAGE Handbook of Current Developments in Grounded Theory* (2019), 145–166.

[96] Chandra Mukerji. 1989. *A Fragile Power: Scientists and the State*. Princeton University Press.

[97] Joel Novek. 2002. IT, gender, and professional practice: Or, why an automated drug distribution system was sent back to the manufacturer. *Science, Technology, & Human Values* 27, 3 (2002), 379–403. https://doi.org/10.1177/016224390202700303 SAGE Publications.

[98] Wanda Orlikowski and Susan Scott. 2008. Sociomateriality: Challenging the separation of technology, work and organization. *The Academy of Management Annals* 2, 1 (2008), 433–474.

[99] Julian E Orr. 2016. *Talking About Machines: An Ethnography of a Modern Job*. Cornell University Press, Ithaca.

[100] Mathieu O'Neil, Laure Muselli, Mahin Raissi, and Stefano Zacchiroli. 2020. 'Open source has won and lost the war': Legitimising commercial–communal hybridisation in a FOSS project. *New Media & Society* (2020), 1461444820907022.

[101] Elena Parmiggiani. 2017. This Is Not a Fish: On the Scale and Politics of Infrastructure Design Studies. *Computer Supported Cooperative Work (CSCW)* 26, 1 (2017), 205–243. https://doi.org/10.1007/s10606-017-9266-0

[102] Eric Raymond. 1999. The cathedral and the bazaar. In *Readings in Cyberethics*, Richard A. Spinello and Herman T. Tavani (Eds.). O'Reilly Press.

[103] RedHat. 2020. The State of Enterprise Open Source. https://www.redhat.com/cms/managed-files/rh-enterprise-open-source-report-detail-f21756-202002-en.pdf

[104] David Ribes. 2014. Ethnography of scaling, or, how to a fit a national research infrastructure in the room. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*. 158–170.

[105] David Ribes and Thomas A Finholt. 2007. Tensions across the scales: planning infrastructure for the long-term. In *Proceedings of the 2007 International ACM Conference on Supporting Group Work*. 229–238.

[106] David Ribes and Thomas A Finholt. 2009. The long now of infrastructure: Articulating tensions in development. *Journal of the Association for Information Systems (JAIS)* (2009).

[107] David Ribes, Steven Jackson, R. Stuart Geiger, Matthew Burton, and Thomas Finholt. 2013. Artifacts that organize: Delegation in the distributed organization. *Information and Organization* 23, 1 (2013), 1–14.
[108] David Ribes and Charlotte P Lee. 2010. Sociotechnical studies of cyberinfrastructure and e-research: Current themes and future trajectories. *Computer Supported Cooperative Work (CSCW)* 19, 3-4 (2010), 231–244.
[109] Dirk Riehle, Philipp Riemer, Carsten Kolassa, and Michael Schmidt. 2014. Paid vs. Volunteer Work in Open Source. In *Proceedings of the 47th Hawaii International Conference on System Sciences*. 3286–3295. https://doi.org/10.1109/HICSS.2014.407
[110] Daniela K. Rosner. 2014. Making Citizens, Reassembling Devices: On Gender and the Development of Contemporary Public Sites of Repair in Northern California. *Public Culture* 26, 1 (2014), 51–77. https://doi.org/10.1215/08992363-2346250
[111] Andrew L. Russell and Lee Vinsel. 2018. After Innovation, Turn to Maintenance. *Technology and Culture* 59, 1 (2018), 1–25. https://doi.org/10.1353/tech.2018.0004 Publisher: Johns Hopkins University Press.
[112] Bert M Sadowski, Gaby Sadowski-Rasters, and Geert Duysters. 2008. Transition of governance in a mature open software source community: Evidence from the Debian case. *Information Economics and Policy* 20, 4 (2008), 323–332.
[113] Salvatore Sanfilippo. 2019. *The struggles of an open source maintainer*. http://antirez.com/news/129
[114] Trebor Scholz. 2008. Market ideology and the myths of Web 2.0. *First Monday* 13, 3 (2008).
[115] Clay Shirky. 2010. *Cognitive Surplus: Creativity and Generosity in a Connected Age*. Penguin UK.
[116] Susan Leigh Star. 1999. The ethnography of infrastructure. *American behavioral scientist* 43, 3 (1999), 377–391.
[117] Susan Leigh Star and Anselm Strauss. 1999. Layers of silence, arenas of voice: The ecology of visible and invisible work. *Computer Supported Cooperative Work (CSCW)* 8, 1-2 (1999), 9–30.
[118] Anselm Strauss. 1988. The articulation of project work: An organizational process. *Sociological Quarterly* 29, 2 (1988), 163–178.
[119] Anselm Strauss and Juliet Corbin. 1994. Grounded theory methodology. *Handbook of Qualitative Research* 17 (1994), 273–85.
[120] Lucy Suchman. 1995. Making work visible. *Commun. ACM* 38, 9 (1995), 56–64.
[121] Lucy Suchman. 2007. *Human-machine Reconfigurations: Plans and Situated Actions*. Cambridge University Press.
[122] E. Carr Summerson and Michael Lempert. 2016. *Scale: Discourse and Dimensions of Social Life*. University of California Press.
[123] Don Tapscott and Anthony D Williams. 2008. *Wikinomics: How Mass Collaboration Changes Everything*. Penguin.
[124] Nathaniel Tkacz. 2014. *Wikipedia and the Politics of Openness*. University of Chicago Press.
[125] Linus Torvalds and David Diamond. 2002. *Just for Fun: The Story of an Accidental Revolutionary*. Harper Business.
[126] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's Talk about It: Evaluating Contributions through Discussion in GitHub. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. ACM, New York, NY, USA, 144–154. https://doi.org/10.1145/2635868.2635882
[127] José Van Dijck and David Nieborg. 2009. Wikinomics and its discontents: A critical analysis of Web 2.0 business manifestos. *New Media & Society* 11, 5 (2009), 855–874.
[128] Kazys Varnelis. 2008. *Invisible City: Telecommunication*. Actar Barcelona, New York.
[129] Juhani Warsta and Pekka Abrahamsson. 2003. Is open source software development essentially an agile method. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*. 143–147.
[130] Steve Weber. 2004. *The Success of Open Source*. Harvard University Press.
[131] Kangning Wei, Kevin Crowston, U. Yeliz Eseryel, and Robert Heckman. 2017. Roles and politeness behavior in community-based free/libre open source software development. *Information & Management* 54, 5 (2017), 573–582. https://doi.org/10.1016/j.im.2016.11.006
[132] Andrea Wiggins. 2013. Free as in puppies: compensating for ICT constraints in citizen science. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*. 1469–1480.
[133] Susan Winter, Nicholas Berente, James Howison, and Brian Butler. 2014. Beyond the organizational 'container': Conceptualizing 21st century sociotechnical work. *Information and Organization* 24, 4 (2014), 250–269.
[134] Alexey Zagalsky, Carlos Gómez Teshima, Daniel M German, Margaret-Anne Storey, and Germán Poo-Caamaño. 2016. How the R community creates and curates knowledge: A comparative study of stack overflow and mailing lists. In *Proceedings of the 13th International Conference on Mining Software Repositories*. 441–451.

Received June 2020; revised October 2020; accepted December 2020