Collaboration Challenges and Opportunities in Developing Scientific Open-Source Software Ecosystems: A Case Study on Astropy

JIAYI SUN, University of Toronto
AARYA PATIL, Max Planck Institute for Astronomy
YOUHAI LI, Carnegie Mellon University
JIN L.C. GUO, McGill University
SHURUI ZHOU, University of Toronto

Scientific open-source software (OSS) has greatly benefited research communities through its transparent and collaborative nature. Given its critical role in scientific research, ensuring the efficiency of collaboration within development teams has become vital. Earlier research has identified both the challenges and opportunities associated with interdisciplinary team collaboration in developing conventional scientific software, as well as the dynamics of distributed teams in the context of open-source software development. However, it remains unclear whether these challenges are identical and if their solutions can be seamlessly adapted to the context of scientific OSS and its broader ecosystem. Therefore, this study examines the challenges and opportunities for improving the collaboration efficiency in the development and maintenance of scientific OSS, focusing on *interdisciplinary collaboration* and *multi-project collaboration* within the open-source environment. We conducted a mixed-methods case study on *Astropy*, a widely-used astrophysics software ecosystem, including (1) a detailed analysis of the commit history to understand the roles and activities of each contributor; (2) an in-depth investigation of cross-referenced issues and pull requests to identify challenges and best practices for cross-project collaboration at the ecosystem level; and (3) an interview study with core contributors to complement the first two steps, examining their collaborative efforts within an interdisciplinary team and across a multi-project ecosystem. We contribute to the CSCW community by deepening the understanding of collaboration in scientific OSS ecosystems, highlighting practices and challenges both at the individual project level and across the broader ecosystem. Our findings offer insights into managing cross-project interdependencies and propose strategies to address key obstacles in scientific OSS development.

Additional Key Words and Phrases: Scientific software, open-source software ecosystem, Collaboration

ACM Reference Format:
Jiayi Sun, Aarya Patil, Youhai Li, Jin L.C. Guo, and Shurui Zhou. 2018. Collaboration Challenges and Opportunities in Developing Scientific Open-Source Software Ecosystems: A Case Study on Astropy. In *Proceedings of CSCW2025 (Conference acronym 'XX)*. ACM, New York, NY, USA, 31 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Scientific software[1] development refers to the process of building software that is used in scientific disciplines such as biology, physics, geoscience, chemistry, and astronomy [33, 42, 86, 109]. It is crucial for supporting scientists

[1]It is also called science/research/academic software. In this paper, we employ the phrase "scientific software" to maintain consistency.

Authors' Contact Information: Jiayi Sun, University of Toronto; Aarya Patil, Max Planck Institute for Astronomy; Youhai Li, Carnegie Mellon University; Jin L.C. Guo, McGill University; Shurui Zhou, University of Toronto.

Manuscript submitted to ACM

in the computational tasks required for scientific discovery [108, 121, 145]. The growing significance of scientific software development necessitates that stakeholders enhance collaboration efficiency, elevate software quality, and ensure long-term sustainability of the software, defined as "*the ability to maintain the software in a state where scientists can understand, replicate, and extend prior reported results that depend on that software*" [109, 159]. In the past decades, substantial financial resources have been allocated to developing and sustaining scientific software in different countries [11, 15, 18, 20, 21, 31, 173]. Despite the recognized importance and devoted efforts, achieving sustainability is still challenge for today's scientific software communities, especially for those that are large and open-sourced [104].

Collaboration between members of the scientific software community directly impacts the sustainability of large and open-source scientific software. The development team performs ongoing maintenance, quality assurance, adaptability, community support, and knowledge management. These factors collectively contribute to the longevity of the project and the effectiveness of the software in supporting scientific research and applications. In this study, we examine the collaborative aspects of scientific software development within the open-source environment. The characteristics of such software, i.e., in the scientific domain and developed in the open-source context, each pose distinct challenges to collaboration. In particular, it requires *interdisciplinary collaboration* between scientists and software developers. However, the two groups often have different objectives (research vs. development) and backgrounds [93, 129, 147, 155], thereby hindering the software development process [94, 105, 110, 136]. Thus, encouraging efficient collaboration throughout the development process significantly impacts the software's sustainability. Meanwhile, following the open science movement, scientific software teams[2] start to follow an *open-source model*, leading many small scientific software communities to transform into large, diverse open-source ones. Additionally, various interoperable scientific OSS projects[3] form a broader ecosystem that enables large-scale scientific discoveries. A notable example includes the creation of the first black hole image, made possible by over 11 software packages from the scientific Python ecosystem [12]. Thus, maintaining sustainable software demands significant communication and coordination across projects.

Researchers in the CSCW and related fields have investigated various collaboration scenarios in the context of scientific discovery, with a particular emphasis on social dynamics, such as cultural differences [89] and the impact of collaboration on scientific outputs [62]. One prominent area of focus is the collaborative development of scientific software, where scientists lacking formal software engineering training work together to build software systems. This type of collaboration introduces several challenges, including the misalignment between academic incentives and the sustainability of software, the undervaluation of so-called "extra work" during software development (e.g., documentation, user support, etc.) [160]. Removing communication barriers among diverse stakeholders and building tools and infrastructure that simultaneously support scientific workflow and promote interoperability and collaboration are also needed [46, 56, 72]. Meanwhile, *interdisciplinary collaboration* between software developers and scientists has also been a subject of exploration when the software becomes more complex and needs better maintainability in traditional scientific software development. Much of this research has centered on in-lab or commercial settings [129, 147], with a focus on team dynamics, workflows, and the motivations and incentives driving software creation. This work highlighted that additional effort is required to transition software into an OSS project that serves as a community tool. However, questions remain concerning what the collaboration practices are in supporting and maintaining existing scientific OSS projects, how collaboration is conducted in settings where interdisciplinarity meets open source – scientists

[2] "Teams" refer to groups of contributors, developers, or maintainers collaborating on software projects.
[3] "Projects" refer to specific software initiatives or codebases within the broader software ecosystem. For example, in the context of the OSS ecosystem, an individual software package or tool is considered a separate project.

and software developers form a vast and diverse group, and they primarily collaborate asynchronously and remotely. Therefore, we aim to explore the question :"**What are the major challenges when interdisciplinary experts collaborate to develop and maintain scientific OSS?**" This question emphasizes *collaboration within individual projects*, where interdisciplinary collaboration is central to overcoming challenges in the development and maintenance of scientific OSS. It highlights the **intra-project** context, where team members from diverse backgrounds work together to achieve shared goals.

On the other hand, researchers from the software engineering field have studied the sustainability of multi-project ecosystems [52, 116, 117] and OSS ecosystems [76] across various topics like processes, challenges, health measures [163], and evolution [80]). They predominantly focused on code dependencies [54, 114, 158], neglecting the examination of collaboration among contributors from different projects. Additionally, CSCW researchers have highlighted the challenges of developing scientific OSS tools, noting that such work often demands additional effort from scientists compared to traditional OSS. For instance, Trainer et al. identified the lack of support for activities like community management, documentation, user support, and training, which are undervalued and under-resourced, thereby hindering OSS projects' long-term sustainability [160]. Similarly, there is a disconnect between academic incentive structures and the effort required for sustainable OSS development: the publication-centric model in academia often overlooks and under-rewards the "extra work" necessary to ensure software contributions benefit the broader scientific community [96, 160]. Despite the growing importance of scientific software development within open-source ecosystems, no prior research has examined these ecosystems where multiple projects are interdependent, sharing knowledge, contributors, and infrastructure. To address this gap, we would like to understand "**What are the intentions and the corresponding challenges in cross-project collaboration?**", as this question focuses on understanding the motivations behind cross-project collaboration and the obstacles it entails, thereby addressing broader ecosystem dynamics that influence the sustainability and innovation of scientific OSS. Investigating cross-project collaboration extends the scope of existing research on within-project dynamics, providing a more comprehensive understanding of collaboration in scientific OSS ecosystems.

Given the diversity of scientific disciplines, it is challenging, if not impossible, to identify a group of software projects that represent the entire landscape of scientific OSS and their communities. Thus, rather than targeting generalizability across all scientific fields, we performed a case study [154], focusing on a specific scientific OSS ecosystem to investigate their collaboration practice. We chose the *Astropy* ecosystem [32], a software ecosystem comprised of 51 interoperable packages that provide essential Python functionalities for astronomy. Its core package (*astropy/astropy*) has been forked over 1.7K times, underscoring its widespread use. Further details are provided in Sec. 3.1. Although *Astropy* has been successful as a scientific OSS ecosystem, ensuring long-term sustainability remains an ongoing pursuit [132]. Hence, we aim to explore existing practices, alongside the challenges and opportunities currently present. In particular, we ask the following two research questions (RQs) that investigate complementary aspects of collaboration within the Astropy ecosystem, focusing on both intra- and inter-project dynamics:

RQ1: (within-project collaboration) What are the major challenges when interdisciplinary experts collaborate to develop and maintain scientific OSS in the Astropy ecosystem?
RQ2: (cross-project collaboration) What are the intentions and the corresponding challenges in cross-project collaboration within the Astropy ecosystem?

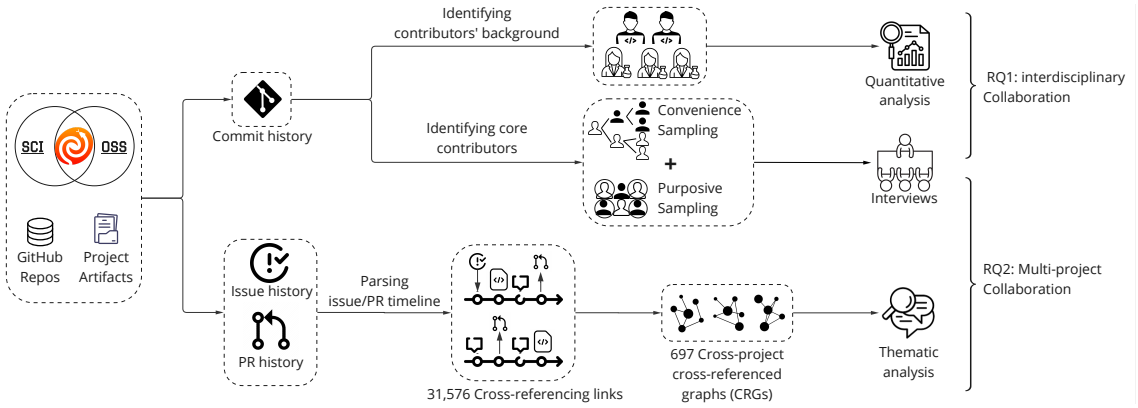


Fig. 1. Research Method Overview.

We adopted a *mixed-methods design* approach [74, 88], as depicted in Figure 1, to explore the challenges inherent in multi-disciplinary and multi-project collaborations, and their potential impact on the sustainability of a scientific OSS ecosystem. Our research method has two components: mining software repositories (MSR) and an interview study. In particular, Our study focuses on the *commit history* to examine contributors' backgrounds and team composition. We then analyze the *timeline of PRs and issues* to identify instances of cross-project collaboration, aiming to understand the motivations behind such collaborations and to identify opportunities for improving collaboration efficiency within the ecosystem. At the same time, *interviews* are conducted with *Astropy*'s core developers from diverse backgrounds to gain insights into their collaboration practices and challenges from both aspects.

Our findings show that the *Astropy* communities face fewer challenges than traditional scientific software in terms of communication difficulties arising from interdisciplinary collaboration. Contributors typically possess some level of knowledge in both fields, and there are experts capable of bridging these interdisciplinary gaps. However, tensions persist between individuals with differing mindsets—those prioritizing scientific objectives versus those prioritizing software development. From an ecosystem viewpoint, beyond the usual collaboration between projects due to code dependencies and efforts to prevent breaking changes, other collaboration activities are also prevalent, including coordinating centralized infrastructure management, common functionality, and knowledge sharing. Drawing from our findings, we propose best practices, future tools, and research directions to foster efficient collaboration and sustained development for the *Astropy* ecosystem and scientific OSS in general.

In summary, we contribute (1) A mixed-method case study that investigates the multifaceted collaboration challenges within a scientific OSS ecosystem from various angles; (2) An analysis comparing the collaboration challenges of modern scientific OSS ecosystems within the broader context of CSCW research, highlighting both unique and shared characteristics with other collaborative phenomena; and (3) Actionable suggestions for future CSCW research and scientific OSS practitioners to advance collaboration in scientific OSS development, with a focus on improving tooling support to foster more effective collaboration and long-term sustainability.

2 Related Work and Background

2.1 Collaborative scientific discovery and corresponding software development

2.1.1 ***Collaboration in scientific discoveries.*** As research questions grow more complex, individual expertise often falls short, encouraging scientific efforts to shift towards more collaborative endeavors [128]. Researchers now have become more specialized in a specific domain [113, 128], making it necessary to collaborate to integrate different areas of expertise [128, 150]. Such trends also foster interdisciplinary collaboration, where the merging of knowledge from various fields leads to new ideas, the integration of diverse insights, and the emergence of entirely new research domains [128]. For instance, researchers have investigated both social and technical aspects of scientific collaboration, such as team composition [59], collaboration productivity [77], knowledge diffusion and sharing [55, 125, 150]. Moreover, through analyzing the citation network, Dong et al. [67] show a shift from individual work to team-based efforts, reflecting the increasing importance of cooperation in science.

Despite these advances, significant challenges continue to hinder collaboration efficiency. These include difficulties in establishing common ground, inconsistent research practices, cultural differences, inefficient communication, and intellectual property concerns [89]. Correia et al. [62] specifically investigated the influence of collaboration types and geographic distribution on scientific output from a CSCW perspective, highlighting how these factors shape the effectiveness of collaboration.

To address such challenges, researchers have leveraged existing tools such as video conferencing to bridge geographic gaps and shared workspaces to facilitate coordination [111]. At the same time, platforms and infrastructures are proposed to better manage the scientific workflow and support collaborations at scale [149]. More advanced solutions include systems for provenance tracking [172], workflow composition, and execution [125], which streamline research processes and improve traceability [112]. In particular, recent research has explored data-sharing platforms, collaborative writing tools, and virtual research environments (VREs) to provide a shared online space for researchers to collaborate [103]. These systems enhance workflows, manage data ownership, and support collaborative knowledge creation by providing shared access to resources [103, 111].

Our study is within the broader context of supporting scientific discovery, focusing on a critical instrumentation of scientific collaboration: scientific software and its development process. We are specifically interested in the scientific software within the OSS context as increasing numbers of scientific software is developed using the OSS model and OSS provides a unique form of interdisciplinary collaboration among researchers as well as between researchers and software developers.

2.1.2 ***Collaborative scientific software development.*** During the collaborative scientific discovery process, scientific software, which acts as both an instrument and an outcome of the process, becomes a critical component that influences how research and scientific collaboration are being conducted [96, 100, 103]. Similarly, the development of scientific software itself is inherently a collaborative endeavor.

The development of scientific software is inherently collaborative, involving contributors with diverse roles and expertise across different stages of the process. In the early stages, researchers typically take the lead, writing scripts and programs to iteratively create an initial version of the software. This phase often involves collaboration within the research team, where junior scientists learn software development practices and domain-specific knowledge by observing and assisting senior colleagues, fostering skill development and expertise transfer [100]. Furthermore, as scientific discovery increasingly involves interdisciplinary collaboration, effective software development often necessitates the

involvement of scientists from diverse fields [38, 96]. Scientific software is often observed to be challenging to maintain and unsuitable for long-term [39, 106, 140] , partially due to the problematic documentation [126, 144, 146] and lacking software engineering training for scientists [164]. Consequently, as software systems grow in scale and complexity, research teams frequently rely on professional software engineers to improve software quality and maintainability. At this stage, collaboration between scientists and engineers becomes critical [105], with scientists contributing domain expertise and specifying requirements, while engineers focus on technical implementation [96, 103]. However, this collaboration poses challenges due to differing priorities– such as balancing long-term maintenance with immediate research objectives–and the demand for specialized skills beyond foundational knowledge and experience in software development [43], which can result in tensions within the development and collaboration process [96].

Beyond the challenges due to the interdisciplinary nature, the academic reward systems also brought tension to the social dynamics during scientific software development. For example, prior work in CSCW studied the incentives and motivations of scientists developing scientific software and revealed that the academic incentive structure often fails to adequately reward software contributions, which can result in the over-production of independent software packages and hinder the development of collaborative projects where researchers can build upon existing work [96]. To address this concern, researchers proposed a system, 'CiteAs', to improve incentives for high-quality software work [68]. The current publication-centric model in academia often fails to acknowledge and reward the engineering work to make software contributions valuable to the broader scientific community: essential tasks like debugging, testing, and user support frequently go unrecognized, leading to their devaluation within academia, threatening the sustainability of scientific software development [130]. Another challenge is the conflicting priorities when it comes to software maintenance. While individual scientists are motivated to develop software for their immediate needs, the lack of incentives for long-term maintenance often leads to software being neglected and ultimately unusable [160]. Additionally, the competitive nature of scientific research can hinder collaboration. Scientists may hesitate to share software or data, fearing that competitors might exploit their work without proper recognition [97, 160].

Building on prior studies from both CSCW and broader research communities that explore the challenges of scientific software development, our work extends this focus by addressing contemporary practices and obstacles. In addition, with the rise of open science [70], an increasing number of scientific software projects adopt the open-source model, introducing greater complexity to interdisciplinary and distributed team composition. Thus, collaboration now extends beyond individual teams to include cross-organizational interactions, as we will elaborate in the following paragraph (Section 2.2). By considering these factors, our study complements existing research by investigating collaboration challenges both within and across projects, providing updated insights into the current state of scientific software development, and offering recommendations to enhance productivity and collaborative efficiency.

2.2 Collaboration in OSS and OSS ecosystem

Following the Open Science Initiative [70], scientific efforts have been encouraged to be more transparent. Scientists develop and share software for research purposes through the open-source model. While the Open Science initiatives promote software and data sharing and reuse, research in CSCW described the potential new challenges it might introduce to scientific collaboration. Here in the following sections, we first examine prior work on OSS and its ecosystems broadly, followed by a focus on research specifically addressing scientific OSS.

2.2.1 ***General OSS and OSS ecosystem collaboration*** OSS projects are characterized by an open development model, where source code is publicly accessible, allowing individuals to contribute freely [45, 63]. Prior work in CSCW has

investigated the social-technical aspects of OSS communities, including the motivations for contributing to OSS where the reasons ranging from altruistic intentions and personal learning to skill development, recognition, and the satisfaction of supporting valuable projects [66, 170]; the barriers for new-comers to join (such as technical hurdles for understanding codebase, difficulty in finding mentors, or cultural differences with the communities etc.) [151–153]. Additionally, existing contributors could also face challenges, including the labor-intensive work of community management [79], navigating toxic online interactions [73, 122], resolving conflicts [99], and managing the risk of maintainer burnout [135]. As OSS projects expand and grow more complex, sustainability concerns, such as funding models and organizational influences, have become critical factors for ensuring the long-term survival of OSS initiatives [78, 79, 81].

Beyond individual OSS projects, often multiple OSS projects form larger ecosystems, such as the Eclipse ecosystem [16], R ecosystem [13, 14], scientific Python ecosystem [90], and the *NPM* ecosystem [165]. Cross-project collaboration involves complex contextual interactions that manage tasks, activities, and interdependencies across multiple projects [60]. Prior research in CSCW has been focusing on the social-technical aspects of OSS ecosystems: the various OSS communities of the projects are viewed knowledge-sharing ecosystems. This ecological perspective highlights the interdependence of developers, users, and the software itself in a dynamic system where knowledge flows are crucial for the community's survival and evolution [161].

Several systematic mapping studies have been conducted on the existing papers on OSS ecosystems from a software engineering perspective, highlighting various areas of future investigation, such as the licensing model, the co-evolution, co-creation, and the market-places [41, 76, 117]. When examining cross-project coordination, the majority of the studies concentrated on the source code dependency (e.g., cross-project bug fixing [54, 57, 114], breaking source code changes [49], software supply chain [107, 158], and the security risks [167]), and the practices of cross-project code reuse [82]. Additionally, various methods were designed to improve coordination efficiency such as impact analysis of cross-project bugs and detecting the affected downstream modules [115], and dependency management tools to improve the quality of the dependency network [91], etc. Moreover, researchers also leveraged social network analysis to illustrate the relationship among developers and projects aiming to support knowledge collaboration across projects in the open-source settings [92, 127].

Unlike previous research, our study examines cross-project collaboration in scientific OSS development, emphasizing the added complexities that arise from its integration with scientific collaboration and its impact on community dynamics. We aim to identify the practices and unique challenges faced by scientific OSS communities within the specific context of a concrete scientific OSS ecosystem. The insights gained from this study will contribute to a deeper understanding of scientific collaboration, supporting open science initiatives by identifying areas for improvement and fostering more effective collaboration practices.

2.2.2 ***Scientiifc OSS and ecosystem*** There has been a growing recognition of the significance of scientific OSS. Numerous funding initiatives have been launched to support maintenance, infrastructure, and community engagement in scientific OSS, along with promoting best practices in software development. For example, the Chan Zuckerberg Initiatives (CZI) established the "Essential Open Source Software for Science" program [101], and the Sloan Foundations funded the "Better Software for Science" program [40]. Similarly, long-term commitment initiatives are created to support scientific OSS from institutions, such as NASA's Open Source Science Initiative [27]. Organizations such as Better Scientific Software (BSSw) [51] and Research Software Alliance (ReSA) [28] have been established to promote best practices in scientific software development and provide training and resources for scientists to improve their software development skills; NumFOCUS [25] has been dedicated to locating financial support for open-source scientific

software projects. Practices like citation support on coding platforms such as GitHub [17] and publishing papers about research software (e.g., Journal of Open Source Software (JOSS) [23]) have been established to acknowledge and give credit to the contribution of scientific software in current academic systems. Despite these efforts, the critical need for high-quality, sustainable scientific OSS continues. Additionally, there has been a drive for community-led actions to advance the development of research software, promoting the efficient sharing, curation, and sustained management of research software artifacts and related knowledge, such as US-RSE [37] and ADORE [34]. These initiatives also seek to enhance support for research software engineers.

Mosconi et al. [124] observed that researchers from various disciplines may have different understandings of data organization, storage, and sharing, requiring careful consideration of these diverse practices, which brings extra workload when committing to the Open Science initiatives [123]. Additionally, turning the software project into a scientific OSS tool often requires extra work from scientists, Trainer et al. described the lack of support for the "extra work" involved in scientific OSS development: community management, documentation, user support, and training are often undervalued and under-resourced, hindering the long-term sustainability of OSS projects [160]. Furthermore, there is a mismatch between academic incentive systems and the effort required for sustainable OSS development: the current publication-centric model in academia often fails to acknowledge and reward the “extra work” necessary to make software contributions valuable to the broader scientific community [96, 160].

Apart from the scholarly efforts from CSCW communities and SE communities, scientific OSS communities publish papers to disseminate their insights into community management, including challenges and sustainable practices. These publications not only enhance the visibility of the software but also promote citations, providing academic recognition for contributors within the reward system. For example, the *NumPy* [26] community published a paper describing its design, use cases, contributors, and funding composition [90], similar to other projects such as *Biopython* [10], *SciPy* [30], and *rOpensci* [29]. Other communities publish technical reports to discuss sustainability challenges focusing on funding, leadership, and maintenance [69, 139]. However, these articles only studied the existing community members and were mainly from a management perspective. Our research contributes to this understanding by exploring sustainability challenges within the context of collaborative scientific OSS development.

In summary, while prior CSCW and relevant research have explored collaborative practices in traditional scientific software development, the collaboration in the context of OSS development mode remains understudied, such as a more nuanced understanding of the social and technical factors that influence scientists’ willingness to contribute to community building, code maintenance, user support, and other activities that go beyond the initial development phase. Additionally, compared with traditional OSS ecosystems, scientific OSS ecosystems are unique in that projects are often interdependent, sharing knowledge, contributors, and infrastructure, yet they might also face additional challenges of interdisciplinary collaboration of scientific software development, such as diverse expertise, communication barriers, and conflicting goals. These dynamics affect both how individual projects operate and how they interact within the broader ecosystem. Consequently, investigating scientific OSS from both within-project and cross-project perspectives is crucial to filling this gap, as it reveals how these ecosystems balance collaboration, innovation, and sustainability in the face of these challenges.

3 Research Design

Our goal is to understand the collaboration challenges and opportunities within the scientific OSS ecosystem from two complementary perspectives: (1) within-project collaboration, with an emphasis on interdisciplinary collaboration (RQ1), and (2) cross-project collaboration at the ecosystem level (RQ2). We adopt a mixed-methods case study [74],

including mining software repositories (MSR) and interviews to both qualitatively and quantitatively understand the sustainability practices and challenges of *Astropy* ecosystem (further details about the project are provided in Sec. 3.1). We adopt an iterative approach to refine the study design, where the data collection, analysis, and results of these methods mutually inform one another [141, 166, 171] (see Figure 1). For example, *MSR* serves many purposes during the study, including determining each contributor's contribution type and selecting interview participants based on commit history (RQ1), identifying cross-project collaboration from issue and pull request timeline events (RQ2), and cross-verifying and enhancing interview findings (RQ1&2). The interview design material and information on cross-reference links analysis are available in the replication package [36], and were approved by the research ethics board of the authors' institution.

3.1 Study Subject: the *Astropy* Ecosystem

The large scientific Python ecosystem. For the choice of the case subject, we leverage the existing structure of the scientific Python ecosystem depicted by previous research [90], in which NumPy [26], SciPy [30], and Matplotlib [24] serve as the foundation of the whole ecosystem, and the rest of scientific OSS is categorized into three layers: (1) *Technique-specific*: scikit-learn, scikit-image, pandas, statsmodels, NetworkX; (2) *Domain-specific*: Astropy, Biopython, NLTK, QuantEcon, cantera, simpeg; and (3) *Application-specific*: cesium, FiPy, yellowbrick, MDAnalysis, etc. We aim to select a project within the domain-specific category that fulfills the three specified aspects. We consider the domain, maturity, popularity, available development artifacts as well as the surrounding local ecosystem of each software as the criteria for the selection. We compare these projects in both *domain-specific* and *application-specific* categories based on four dimensions: project age, commit count, contributor count, and star count, as shown in Fig. 2. Eventually, we choose *Astropy* [7], an ecosystem of software packages[4] for astronomy, as the subject because it is a popular and mature project with the largest number of contributors and commits. With over 18,461 citations of the papers published by the community [131, 133, 134] over 10 years, *Astropy* is widely used by researchers and practitioners in astrophysics research as well as scientific missions, such as James Webb Space Telescope [22] and Event Horizon Telescope Collaboration [12, 19]. The code repositories and development artifacts of the packages in the *Astropy* ecosystem are publicly available via GitHub and official documentation. This allows us to study tangible issues in a rich context related to sustaining software projects, fostering communities, and enhancing collaboration efficiency throughout the ecosystem.

The community. Community members could be divided into: (1) the official *Astropy Team*, comprising individuals with administrative roles, involving tasks related to code contributions (e.g., maintenance, development&operation) or non-code contributions (e.g., handling finances, community engagement); and (2) contributors, who do not hold official roles but actively contribute to the repositories through various means (e.g., code contributions, reporting issues).

The ecosystem. *Astropy* is an ecosystem including 51 computing-related Python packages (as of August 2022) and multiple infrastructure packages [133]. As of August 2022, the *Astropy* ecosystem comprises 51 computing-related Python packages, further categorized into three groups:

- The *core* package, *astropy/astropy*, provides common functionalities such as data transformations, basic computations, and logging system [4]. It is maintained by "core-package maintainers" [5], who are part of the official *Astropy Team*.

[4]To avoid confusion, we will use "package" throughout the paper, as it is often used interchangeably with "project."

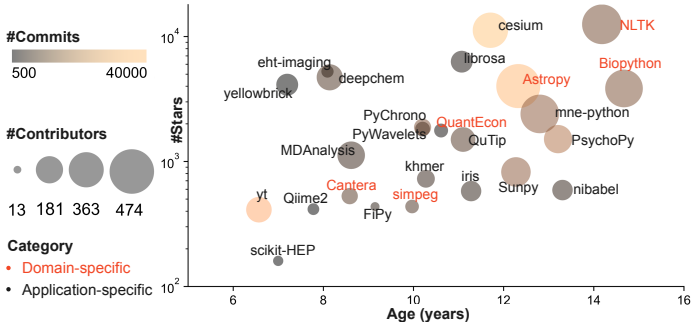


Fig. 2. The scientific Python ecosystem. For each dot, the size represents the number of contributors to the package, the color gradient indicates the commits count of the package, and the color of the text labels demonstrates the category of the packages.

- 7 *coordinated packages*, that are either "experimental or problem space-focused" not fitting into the core package at that moment [3]. They are developed and maintained by the "coordinated package maintainer", also part of the offical *Astropy Team*.
- 43 *affiliated packages*, are astronomy-related packages offering functionalities that are not part of the core package. These packages are not under the management of the official *Astropy Team*; rather, they are maintained by distinct teams, forming diverse sub-communities within the ecosystem [2].

According to the official project documentation [7], the code base of the packages is continually evolving. For example, if a functionality within a *coordinated package* becomes widely used across numerous cases, the code may be incorporated into the *core package*. Likewise, if an *affiliated package* proves valuable to a broader community, the *official Astropy Team* may take over the maintenance responsibility and elevate it to a *coordinated package*. Over the past 10 years, the *Astropy* ecosystem has transformed from a single core package *astropy/astropy* and progressively grown into a comprehensive ecosystem, consisting of various interoperable packages that either build upon or extend the functionalities of the core package [133]. With the goal of promoting code reuse and collaboration, *affiliated packages* joined the ecosystem after being reviewed by the *official Astropy Team* members and meeting the requirements of the ecosystem guideline.

Apart from the 51 computing-related packages, there are several *infrastructure packages*, maintained by the *official Astropy Team*, that assist sub-communities within the ecosystem in easily building and maintaining their packages, lowering the barrier for scientists with little SE experience [133]. These infrastructure packages consist of pre-configured templates, build tools, documentation generation, and testing toolkits [6].

Sustainability challenges reported by the community. While Astropy stands out as a notably successful scientific OSS ecosystem, having implemented many best practices including fostering equity, diversity, and inclusion (EDI) through workshops, securing financial support through various grants, providing unified infrastructure to lower contributors' workload, the community still faces significant challenges. These include sustaining an engaged community, managing inevitable turnover, recruiting SDEs with specific domain expertise, and handling heavy maintainer workloads [132]. Building on these insights, our study aims to explore challenges related to collaboration, complementing existing knowledge, with the goal of generating practical recommendations to enhance the community's long-term sustainability.

3.2 Assessing team composition regarding interdisciplinary expertise through commit history (RQ1).

We analyzed the commit contributions of all contributors to the core package to approximate their backgrounds. According to the project documentation, the files in the code base can be grouped into two types: (1) *science-related files* (e.g., source code of domain-specific functions, test cases, and documentation), and (2) *engineering-related files* (e.g., config file, utilities such as logging, or code of conduct), which are perceived as infrastructure&management related. For each contributor, we compute the *ratio of science-related contribution* summing up the lines of code (LOC) contributed in science-related files over the entire commit history and then dividing it by their total number of contributed LOC.

3.3 Interviews with core contributors to understand collaboration challenges and opportunities (RQ1&2).

Given the pivotal position of the *core* package within the *Astropy* ecosystem, around which most other packages revolve, we consider core contributors of *astropy/astropy* as key informants, as they have long-time involvement in the *Astropy* ecosystem, play a key role in its communities' evolution, and would have an in-depth view of these topics on understanding the challenges surrounding within-project *interdisciplinary collaboration* (RQ1) and the *cross-project collaboration* at the ecosystem level (RQ2). Furthermore, as many core contributors to *astropy/astropy* also serve as maintainers or core contributors for affiliated and coordinated packages within the ecosystem, this enables simultaneous access to insights about other sub-communities.

Recruitment. We identify 41 core contributors from the 424 contributors, considering their substantial contributions to the source code (responsible for 90% of the total commits) similar to prior works [61, 64]. We then sent interview invitations using the email addresses collected from their public profiles. Meanwhile, we applied *convenience sampling* [120, 137, 148] by requesting the interviewees to recommend other core contributors. Furthermore, we aim to include contributors from diverse backgrounds, such as researchers and professional software engineers, to gather comprehensive insights on various aspects (known as *maximum variation sampling* [156]). In total, we conducted 11 semi-structured interviews (referenced as P1–P11 in the results when citing interviewees). These 11 interviewees' code contributions covered 40 packages within the Astropy ecosystem, including the core package, 32 affiliated packages, and seven coordinated packages.

Interview protocol design. The interviews were done remotely on Zoom except for one in person, all lasting 45 to 60 minutes. The interview script underwent iterative revisions, involving the removal of irrelevant questions, the addition of new ones, and improvements to existing ones. There was no compensation for participating in the interview study. During the interview, we asked participants about their roles in the community, their educational background in both SE and the scientific domain, their experiences in contributing to OSS projects, their opinions on interdisciplinary and multi-project collaboration in the *Astropy* ecosystem and sustenance of its community and the corresponding ecosystem, etc.

Analysis and participants validation. We conducted our analysis using a thematic analysis approach [65]. Each interview was transcribed and analyzed immediately after completion. As the analysis progressed alongside recruitment and new interviews, the results from earlier interviews helped to inform the follow-up questions in later ones, enabling us to gather in-depth insights and richer context from the interviewees. In particular, for each interview, two authors first independently analyzed the transcript using an inductive open-coding approach to identify emerging themes [142]. Following the initial coding, the authors discussed their findings and reached a consensus on the identified themes. To ensure the accuracy and credibility of our results, we sent follow-up emails to the interviewees soon after the analysis of their interview transcripts to clarify any ambiguous information and to validate the identified themes [53]. Through this iterative process of data collection and analysis, our interviewees, as long-term contributors to the team, offered

specific and detailed descriptions of their collaboration process and provided the concrete context of the challenges they faced. We noticed that after the ninth interview, the subsequent interviews offered only marginal additional insights; they enriched our understanding of the existing themes, but no new theme was merged from those interviews. We determined that the richness and depth of our analysis was sufficient to answer our research questions [50, 75, 87]. Therefore, we stopped recruiting further interviewees after 11 interviews were conducted.

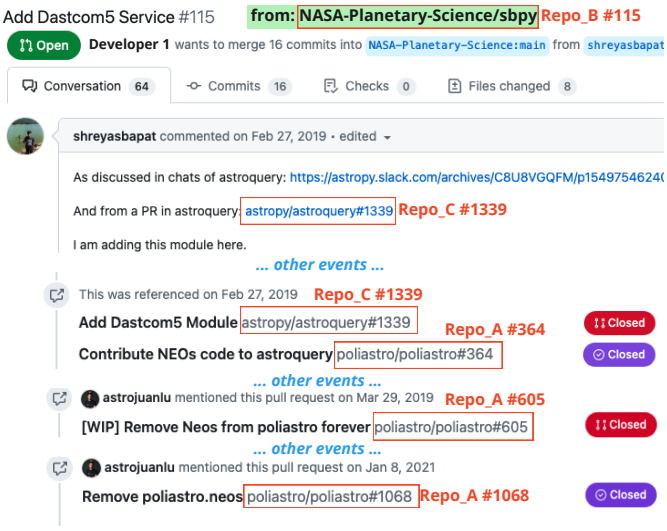
3.4 Examining cross-project collaboration through cross-referenced issues and PRs (RQ2).

To understand the collaboration challenges from the ecosystem perspective, with a particular emphasis on the dynamics of cross-project collaboration, we investigate the interaction among different communities through *cross-reference links* during code-review and issue-resolving processes as instances.

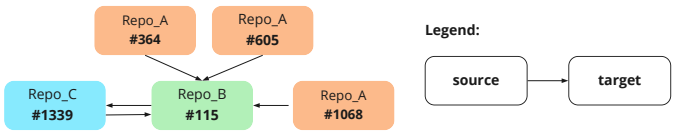
Collecting cross-reference links. GitHub offers cross-referenced links either within or across projects as directional links between a PR/Issue (source) to another PR/Issue (target) created during discussion. Figure 3(a), presents the partial timeline events of *PR#115* in the project of *NASA-Planetary-Science/sbpy*. Initially, it established a cross-reference link to the *astropy/astropyquery* project as a source node upon creation. Subsequently, it was referenced four times as a target from the other two projects (*poliastro/poliastro* and *astropy/astropyquery*). We collected all the cross-referenced links in the *Astropy* ecosystem by parsing all the issues and PRs using the GitHub Timeline events API [1], resulting in a total of 31,576 links. We then excluded links within the same project and retained only those connecting source and target nodes from different projects. If there are multiple links that exist from source to target, we only count one occurrence to avoid duplicated analysis. This results in a final count of 1,491 links across 43 distinct packages.

Graph construction. We construct a *Cross-Reference Graph (CRG)*, where the nodes represent issues/PRs, and edges are cross-reference links between nodes. Note that the graph is not fully connected but includes several disjointed components (also known as sub-graphs), meaning some nodes cannot be reached from other nodes. In total, the 1,491 links contribute to the formation of 697 CRG sub-graphs. In the rest of this paper, we call each "CRG sub-graph" CRG for short. For example, Figure 3(b) illustrates part of a CRG as an exemplar, reflecting the example in Figure 3(a).

Analysis. With the CRGs and the corresponding discussions in the linked issues/PRs, we conducted a thematic analysis [65] to understand the intentions of cross-project collaboration as well as the challenges and practices. Due to the labor-intensive nature of the manual analysis, we focused on a sample of the large-size CRGs to infer the collaboration activities as they involve multiple projects. In particular, we selected the 95% quantile from all CRGs, resulting in 34 CRGs with seven or more nodes and 375 edges. For each CRG, the analysis process consists of the following steps: (1) We read through the title, body, and all activities along the timeline of the linked issue/PRs to understand the contextual information. (2) Then, we systematically explored all the links in a breadth-first manner to examine the reasons behind the connections. (3) We focus on the underlying problem-solving objectives and the overarching topic around the discussions to understand the collaboration effort. In this way, we can infer the reasons for one issue/PR to be linked to another issue/PR in another project within the ecosystem, which is the "collaboration intention" for initiating the cross-project linkage. (4) Further, we document the inferred "collaboration intention" as labels for the cross-reference pairs. It is common to identify multiple labels in one CRG. To reduce the possible bias of the analysis [141], two authors first independently analyzed the 20 links in the CRGs with the steps described above and reached a consensus. Then one author analyzed the remaining CRGs.



(a) Example: Cross-reference events connecting three projects.



(b) Cross-Reference Graph (CRG) for the example above.

Fig. 3. Cross-reference events example and the corresponding Cross-Reference Graph (CRG), where the nodes represent issues/PRs, and edges depict connections between source and target nodes from three projects.

4 Interdisciplinary Collaboration (RQ1)

In this section, we first present the composition of the core contributors in terms of expertise in the scientific domain and software engineering. Next, we delve into the concrete obstacles encountered by our interviewees during their interdisciplinary collaboration.

4.1 Interdisciplinary Team Composition.

As described before, we analyzed the commit contributions of 424 contributors to the core package. The number of commits merged by each contributor varied significantly, ranging from 1 to 4,475 over an 11-year period, with a median of 4. To approximate contributors' backgrounds, we group the files in the code base into *science-related files* and *engineering-related files* according to the project documentation. We visualize the result in Figure 4 , with the left side of the spectrum representing a higher contribution to engineering-related activities, and the right side indicating more science-related contributions.

Interviewees' backgrounds. All 11 interviewees are in the official *Astropy* team, tasked with decision-making responsibilities including community management, maintaining the packages, making long-term plans, etc. They have extensive backgrounds in astronomy, with six holding doctoral degrees, one having completed a master's degree, and two being Ph.D. students in Astrophysics. They are from seven different organizations, such as research institutes or

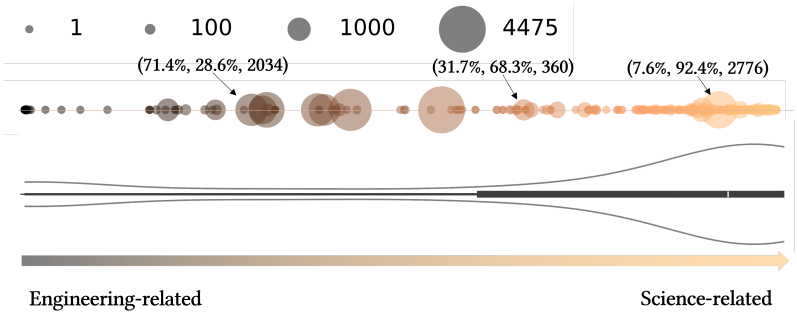


Fig. 4. Composition of contribution types of 424 contributors to Astropy core library. Each dot represents a contributor and size reflects the number of commits. For each dot, $(e\%, s\%, c)$ represents the ratio of *engineering-related* contribution, *science-related contribution*, and *the total number of commits*. The violin plot on the bottom presents the distribution of the number of contributors across different ratios.

universities.[5] Two interviewees were self-identified as software developers and also acknowledged by others as the only few ***professional software developers*** among the decision-making team. They are mainly involved in DevOps, project maintenance, code review, etc. One is paid by an institute to maintain *Astropy*, and the other is contributing voluntarily. The rest of the nine interviewees have daily jobs as researchers and hesitated to label their roles as either software developers or researchers. As P5 pointed out, it is more appropriate to "*think of it as a spectrum from pure researcher to a pure software developer. There are people at all parts of that spectrum*". The statement corroborates the commit history analysis results shown in Figure 4, indicating that contributors possess a diverse range of expertise without a strict demarcation between the roles of scientists and software developers. Meanwhile, they are also ***users*** of some *Astropy* packages during their daily research activities. Five of them are ***co-founders*** of *Astropy*, driven by the desire to establish a unified standard for various tasks, as they found that the development of scientific software in their domain was fractured, with individuals developing their own tools to attain similar goals. Despite lacking formal training in professional software engineering, they have made substantial code contributions to *Astropy*.[6]

4.2 Communication Effectiveness between scientists and software developers.

Our interviewees did not view the barriers between science and software engineering as an issue within the OSS context, considering the core contributor team demonstrates a "*spectrum of expertise*" (P5). As P5 suggested, "*there is always someone you can talk to who understands the domain knowledge of what you are working on but also has a lot of knowledge in the software structure.*" P6 also agreed that in an open-source environment, most of the scientists who have engaged with the group are (at some level) thinking like software developers.

In contrast, interviewees encountered more notable communication challenges in their daily work environment outside of the open-source setting, such as in research institutes and labs, where researchers and software developers within the same organization frequently face communication breakdowns when conveying domain-specific functional requirements for the software. These results align with the findings from prior work focusing on the communication

[5]To prevent the risk of inadvertently revealing the identity of our interviewees, we refrain from providing specific details of their demographic background. The *Astropy* coordination committee has provided consent for disclosing the project name and has reviewed the manuscript. All interviewees are aware of the potential risk of being identified as being part of our study.

[6]Note that there is a discrepancy between the paper published by the *Astropy* team in 2022 [133] and our results regarding the distribution of software developers and scientists in the team. The *Astropy* paper focuses on the "top contributors" ranked by the number of commits which is easier to measure quantitatively, while our interviewees mainly refer to the "decision-making roles" in practice, where only a few software developers are involved.

challenges in the research institutes and co-located labs [93, 94, 105, 110, 129, 136, 147, 155]. As P8 described: "*...in science, there's a culture where you explore and figure it out; in engineering, it's quite the opposite...Scientists would be super frustrated, questioning why engineers keep asking them about what they want done, while engineers wonder why the scientists won't tell them what they want...[The issue often lies] scientists don't really think about requirements.*"

4.3 Tension between Different Mindsets.

Nevertheless, a noticeable discrepancy is observed between the *mindsets* of the scientists and software developers in *Astropy*. One type of tension is regarding ***task prioritization***. Professional software developers believe it is important to follow software engineering best practices and utilize automated workflows, such as Continuous Integration and Continuous Deployment (CI/CD), to reduce maintenance burden. As a result, they often explain to scientists that their code submission does not meet the code quality standard and needs refactoring. However, scientists view software developers as individuals who "*are not as familiar with the kind of flexible nature of scientific software collaboration*" (P5).

When tensions between differing mindsets arise, especially concerning coding standards, scientists in the team react differently. While some scientists are open to learning software engineering best practices, others demonstrate resistance. Those who refuse to adhere to the standards may become "*defensive and then just disappear and leave the pull request there to rot*", as noted by P2. This can substantially burden the software maintenance workload. The resistance could be due to their primary focus on research rather than software development, resulting in limited time for code refactoring to meet contribution requirements. P4 pointed out that scientists tend to "*contribute as needed*" and then "*go off to do other research until that happens again.*" The software developers in such cases have to make the hard choice of either taking over the code and improving it or ignoring the PR. Moreover, several interviewees pointed to long-standing issue discussions on GitHub as evidence of the tension between different mindsets. For example, in a discussion thread [8], one maintainer expressed concerns about the current development infrastructure and its fragility, urging prioritization of maintenance work due to the increasing complexity of the ecosystem. Meanwhile, other contributors raised concerns over the complexity of workflow, which could steepen the learning curve for scientists and discourage their contributions.

Another tension revealed is the ***perception of seniority***. According to one interviewee who has experience in both *Astropy* and other OSS communities, one distinction between these two is the ranking of seniority – in traditional OSS, contributors are ranked by the volume of their code contributions [162], while in *Astropy*, people with a senior academic title could have more decision-making power on whether to merge PRs. Other interviewees also observed that "*senior researchers in the decision-making position sometimes ignore certain PRs because they do not see the value of the research.*"[7] Furthermore, sometimes the contributors who submitted PRs are not informed about the reason for keeping the PR open or rejected, which can be frustrating. One also mentioned that despite the management team's "do-ocracy" policy [133], which aims to empower those actively contributing to the project and ensure fair decision-making, there is still a noticeable difference between individuals with a scientific mindset and those with an engineering focus.

Answer to RQ1: Our findings highlight the unique dynamics of interdisciplinary collaboration within the Astropy ecosystem. While the team benefits from a diverse mix of scientific and self-taught software engineering expertise and experiences reduced communication barriers compared to traditional scientific software development settings, challenges persist. These include tensions stemming from differing task priorities and perceptions of seniority, which influence decision-making processes and collaborative efficiency. Together, these insights underscore both the strengths and ongoing complexities of fostering interdisciplinary collaboration in open-source scientific software development.

[7] To protect the identity of the interviewee, we don't label them on the quote here.

5 Cross-Project Collaboration (RQ2)

In this section, we discuss the identified cross-project collaboration intentions and ecosystem-level challenges for *Astropy*, based on the mixed-methods approach outlined in Section 3. In particular, we investigate concrete instances of the cross-project collaboration that was manifested in issues and PRs and/or mentioned by our interviewees, and the reasons for initiating such collaboration. Finally, we discuss factors reducing the collaboration efficiencies identified through both the interview and qualitative analysis.

5.1 Communication and Collaboration within the *Astropy* Ecosystem

5.1.1 ***How cross-project communication happens.*** Our interviewees mentioned several communication channels used by the community members, including Slack, Facebook groups, and the Astropy Dev mailing list, which has over 1,000 subscribers. At the same time, GitHub serves as one of the primary tools for members to collaborate around the code bases. In particular, the cross-reference feature for issues and pull requests on GitHub automatically links information across different projects, facilitating discussions that involve participants from various communities.

As mentioned in Section 3.4, we have collected 1,491 links between pairs of issues/PRs across 43 distinct packages contributing to the formation of 697 Cross-Reference Graphs (CRGs) (denoted as CRG 1-697). As shown in Figure 5, these CRGs differ in size measured by node count, ranging from 2 to 27 nodes (mean=2.94, SD=2.18, median=2). The number of links ranges from 2 to 32 per CRG (mean=2.14, SD=2.55, median=1). Most CRGs involve 2 different packages (mean=2.26, SD=0.99, median=2). This indicates that although there are cases where multiple issue/PR are discussed together within a common context, the prevailing pattern shows that most cross-project coordination involves two nodes from two different projects connected by one cross-reference link. The detailed nodes and links of each CRG could be found in the replication package [36].

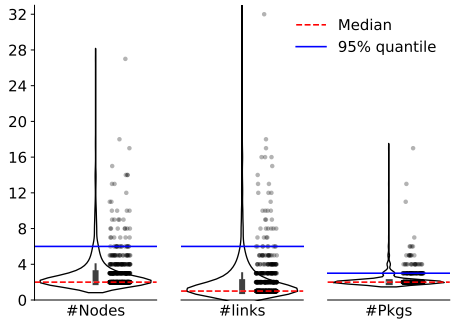


Fig. 5. Violin plot illustrating the statistics of the Cross-Reference Graphs (CRGs)

5.1.2 ***Intention behind cross-project collaboration.*** To understand the reasons for cross-project collaboration, we asked interviewees about scenarios where they need to collaborate with contributors from other projects within the *Astropy* ecosystem and conducted a qualitative analysis of cross-referencing links to derive these reasons. Following the qualitative analysis procedure (described in Sec. 3.4), we identified themes concerning the intention behind the cross-project collaboration. We excluded 9 edges that were mistakenly created by the developers in 3 CRGs. By synthesizing the results from both interviews and CRG analysis, we identified five key themes from 366 edges: (1) Change coordination due to code dependencies, (2) Knowledge sharing, (3) Coordinating shared functionalities, (4) Centralized infrastructure

management, and (5) Coordinating during release cycles, the last of which emerged exclusively from the interviews. We presented the distribution of the five themes in Table 1. Most CRGs have more than one theme identified. In Figure 6, we depict the composition of link types in each CRG.

Table 1. Summary of the intentions behind cross-project collaboration, based on interviews and CRG analysis.

Collaboration Intention	#CRGs	#Links (ratio)	Mentioned by Interviewees
Change coordination due to code dependencies	21	145 (39.63%)	P3
Knowledge sharing	22	97 (26.5%)	-
Coordinating shared functionalities	15	92 (25.14%)	-
Centralized infrastructure management	3	32 (8.74%)	P1-3
Coordinating during release cycles	-	-	P2, P4, P6

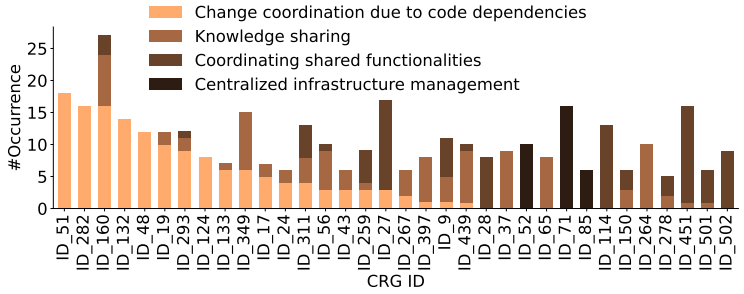


Fig. 6. Distribution of CRG labeling results. Each bin represents one CRG. The different colors indicate the different themes identified in each edge within the CRG. The bins are arranged in descending order based on the number of edges in the "change coordination due to code dependencies" category.

(1) Change coordination due to code dependencies. This type of cross-project collaboration arises from code dependencies between upstream and downstream packages, constituting 39.63% of the links. To gain an overview of the code dependency within the *Astropy* ecosystem, we parse the PyPI installation requirement[8] among the 51 packages to identify their direct upstream. Figure 7 illustrates the dependency relationships among packages within the ecosystem. The results indicate 46 out of the 50 *coordinated* or *affiliated packages* directly depend on the *core package*, with many packages being also interdependent with each other. This phenomenon is akin to previous research exploring the interdependence of projects within a software ecosystem. Because of the interdependent code relationships of the packages in the ecosystem, it is not surprising that resolving dependency-related issues such as version syncing, refactoring as reactions to API changes [95, 138], breaking changes [48, 168], and cross-project bug fixing [114, 115], etc. As P2 confirmed that: "*when things start to break, there was a lot of communication.*"

(2) Knowledge sharing. Cross-project links are also used to facilitate knowledge sharing and improve collaborative discussions. This theme accounts for 26.5% of the links across 22 CRGs. Considering that all packages within the *Astropy* ecosystem fall under the same domain, solutions found for one package could serve as inspiration for others. People from different communities often help each other to better understand the rationale behind the code changes or feature implementation. For example, in CRG_160, contributors from *poliastro/poliastro, issue#222* and *astropy/astropy, PR#3749*

[8]Python packaging metadata for distribution requirement:https://shorturl.at/75XXA

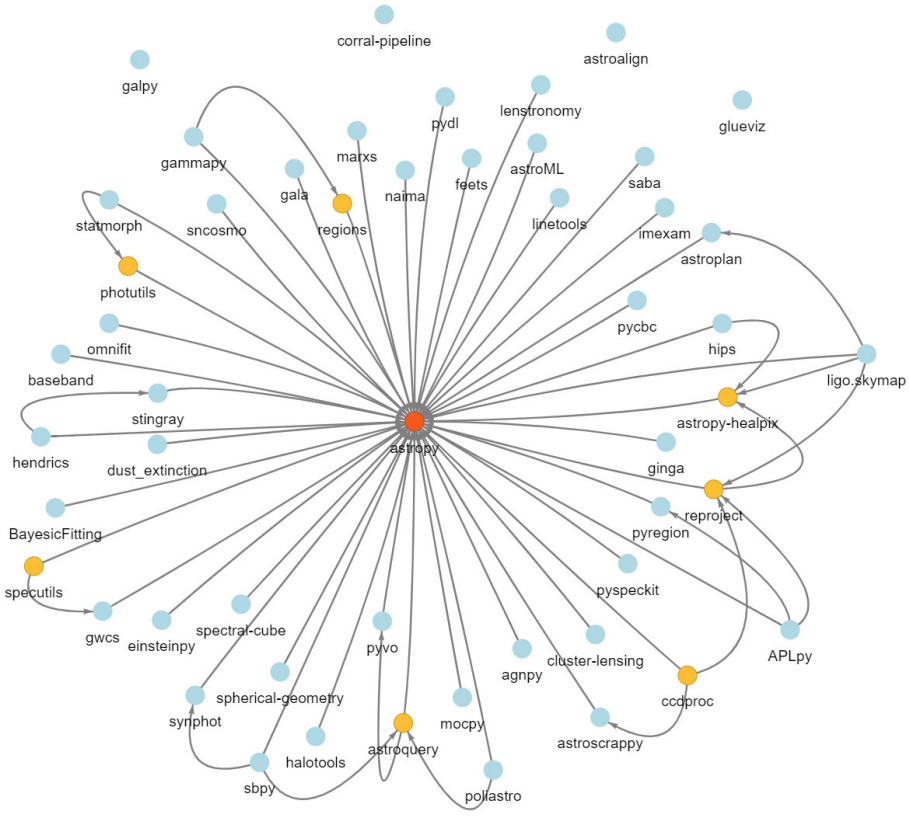


Fig. 7. Code dependency relationships among packages within the *Astropy* ecosystem. The orange node represents the *core* package, yellow nodes represent *coordinated* packages, and blue nodes represent *affiliated* packages. Arrows point from each package to the package it depends on

discuss together to figure out the implementation of the data format (coordinated system reference frames) provided by different organizations.

(3) Coordinating shared functionalities. Contributors from multiple communities also collaborate to develop new features, refactoring, and/or migrating existing features. 15 out of 34 CRGs exemplify situations in which projects collaborated to enhance features to achieve better usability, leading to mutual benefits across multiple projects. Overall, we recognized 25.14% of the links in this category. A typical example is in CRG_28, where the conversation originated from a feature request of "*the ability to select existing observatories for EarthLocation*" (*astropy/astropy, Issue#3813*). The feature was initially implemented in a *coordinated package (astropy/astroplan)*, and ported to the *core project* to benefit a broader audience. Eventually, after one year, contributors from both projects collaborated to eliminate redundant code and incorporate additional test cases, ensuring the feature's overall quality.

(4) Centralized infrastructure management. *Astropy* is a multi-project ecosystem following a centralized management strategy aimed at enhancing sustainability. This strategy enforces pre-configured infrastructure packages that facilitate the maintenance effort for each package. According to interviewee P2, an experienced software developer:

"*We have the infrastructure for the easy stuff to automatically open batch pull requests. So if the error came from our template, we just automatically open pull requests across the package that we know about. And tell them, hey this has changed, you need to accept this change or things will break. Sometimes they do that. If that's not easy, we batch open issues that link them to whatever information they need to apply the patch themselves. So where we can, we automate. If not, we use communication channels like Slack on the mailing list.*"

Moreover, when the infrastructure needs updating or is broken due to dependent package updates, such as PyTest, effective communication becomes crucial. As P2 and P6 mentioned, "*communication was just fired up*," indicating a surge in dialogue regarding the failing infrastructure. This scenario is also observed when maintainers from Astropy engage with projects outside their ecosystem to resolve breaking changes.

The scenario described above by our interviewees can also be observed in the cross-referenced events — 32 links (8.74%) from three CRGs (ID_52, 71, 85) represent the cross-project coordination related to the shared infrastructure supporting the community's software development activities (e.g., testing tools). The cross-reference links in those cases were created by the official *Astropy* team maintainers to broadcast code changes to other packages using the infrastructure. For example, in CRG_52, 11 projects participated in the conversation about updating the code coverage tool that is shared by many packages (*astropy/astropy, PR#12245*). Further, the maintainer of the core package automatically opened 11 PRs using *batchpr* [9] under each repository to patch the affected code blocks.

(5) Coordinating during release cycles. In addition to the collaboration manifested in the cross-referencing events, three interviewees (P2, P4, P6) mentioned the need to proactively coordinate for package release. They noted that several packages follow a six-month release cycle, resulting in increased communication during that period. Furthermore, centralized infrastructure can facilitate easier package releases for scientists, as explained by P2:

"*So we set up a way that scientists who knew nothing about releasing a Python package ... we maintain the package template that came with a lot of infrastructure, already in place, [including] Config files for Travis ... So basically they can set it up and work out of the box releasing and packaging.*"

During the release preparation, it is also necessary to test the corrective measures to ensure they do not disrupt the existing code. Core team members actively contact downstream package maintainers to remind them to run the tests and report back if they see problems (P2). Similarly, P5 described:

"*... anytime we do a release of the core package, every six months now, we have everybody who has an affiliated package test with at least an outreach moment, because we reach out to them have them test the new version of Astropy core, send any feedback about that, engage with us to see if we broke things for them. So there's at least two check-ins per year at that level, but other than that, it's constant chatter in the mailing lists and in Slack.*"

5.2 Obstacles for cross-project collaboration

While *Astropy* achieved significant success and maturity, our interviews and qualitative analysis identified several instances of collaboration inefficiencies that hindered its full potential. This section outlines three specific challenges and inefficiencies.

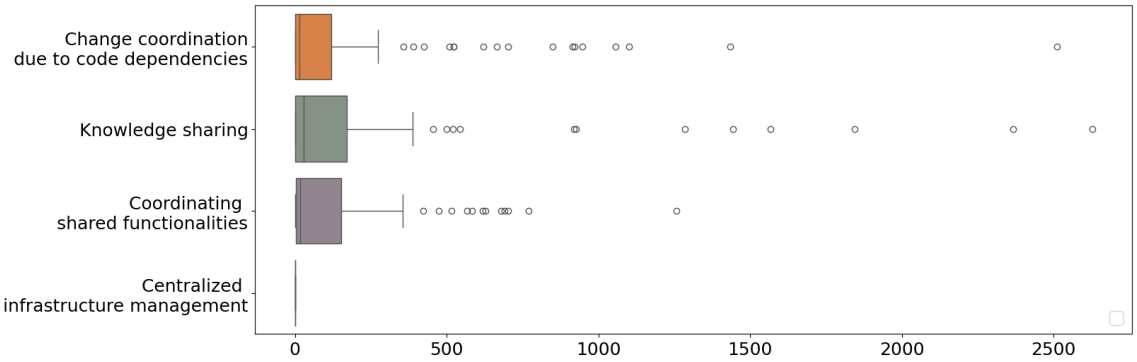
Hard to find relevant information among scattered issue threads. With an increased number of linked issue/PRs, coupled with prolonged discussions, individuals often experience difficulties navigating through multiple cross-linked discussion threads and extracting useful information. The complexity and volume of conversations can overwhelm contributors, making it challenging to maintain a clear overview and keep track of the conversation. Building on prior work highlighting the difficulty of retrieving information from lengthy and intertwined discussion threads focused on a single issue [44, 84] , this challenge becomes even more pronounced when discussions span multiple projects. As

shown before, there were 245 CRGs have at least two links. This necessitates participants to navigate between pages to monitor information. Furthermore, all cross-referenced links were manually created by developers responsible for maintaining oversight of pertinent information. For example, in 2022, an issue (#1345) was raised in the *photutils* project (CRG_37) to address a software bug causing crashes. During subsequent discussions, a developer referenced a related issue from the core package (*astropy, #3575*), which had been created seven years earlier, to propose potential solutions. Given the significant time intervals between discussions on different issues, there is a potential for loss of knowledge and inefficient communication within the ecosystem due to the volume of conversations. It is impractical to rely solely on individuals who have been in the community for an extended period to manually create links and aggregate relevant information.

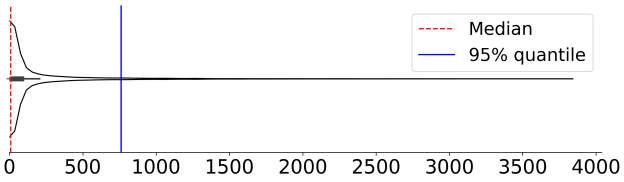
To corroborate this finding, we computed the time gap, measured in days, (denoted as ΔT) between the creation time of the cross-referenced link ($T_{\text{LinkCreation}}$) and the creation time of the target node ($T_{\text{TargetCreation}}$). This analysis aims to estimate the temporal distance of pertinent information and assess potential information loss. Figure 8(a) illustrates the distribution of ΔT for all cross-referenced links. The results indicate a median duration of 8 days for linking source and target issues/PRs. However, some links fall within the 95th percentile, taking up to 761 days to establish. Figure 8(b) displays ΔT for the four link types identified through qualitative analysis. We observe that since the links were created by the core team automatically batch PRs downstream, the link was created when the target was created to provide context information, therefore $\Delta T = 0$. While for the links of *change coordination due to code dependencies*, we found over 37 links (75th percentile) that were created 119 days after the target issue/PRs was created, a maximum of 2510 days; for the links of *knowledge sharing*, we found over 25 links (75th percentile) that were created 171 days after the target issue/PRs was created, a maximum of 2627 days; and for the links of *coordinating shared functionalities*, we found over 23 links (75th percentile) that were created 153 days after the target issue/PRs were created, a maximum of 1256 days. The results underscore the importance of maintaining an overview and aggregating pertinent information in a traceable manner. Without effective traceability, there is a risk of information loss, which can impede collaboration.

Wasted effort and sub-optimal solutions. Divergent requirements from various communities can result in conflicts and sub-optimal solutions. For example, in CRG_451, contributors from three projects (*poliastro*, *sbpy*, and *astroquery*) found that they share similar features with different implementations that could be combined, preventing duplicated efforts. Thus, *poliastro* submitted a PR to *astroquery*, and another PR to *sbpy* with the intention of improving usability and making *poliastro* lighter. However, neither PR was merged in the end. The first PR got rejected by *astroquery* maintainers because the code "*was a subset of already existing functionality without providing much functionality complementary.*" The maintainer in *poliastro* then decided to remove the *NEOs* module entirely since it had become burdensome to maintain and *astroquery* provides a potentially better alternative. They also noted the downside of this move is losing the authorship of the code. The second PR was ignored for over four years and is still pending due to a long period of inactivity in *sbpy* and thus the effort of creating the PR was wasted.

Protracted responses from downstream packages. Despite the ecosystem being centrally managed, interviewees highlighted several instances of inefficient communication. For example, interviewees reported that when core team members automatically open pull requests for downstream packages, it often takes time for these projects to respond and integrate the code changes. P2 stated, "*it's a lot of effort to do that*". Furthermore, within this rapidly growing ecosystem, opening pull requests automatically is just one aspect; P2 pointed out that these pull requests can frequently introduce new issues. This necessitates a back-and-forth process to determine responsibility for fixing the breaking changes. Another example of inefficient communication, as described by P2, occurs when core package maintainers attempt to encourage downstream packages to test their development versions. While they have successfully persuaded



(a) Distribution of ΔT (=$T_{LinkCreation}$ - $T_{TargetCreation}$) of the sampled cross-referenced links of different intentions.



(b) Distribution of ΔT of all the cross-referenced links.

Fig. 8. Distribution of ΔT.

some mature downstream packages to include the development version of Astropy in their CI processes to detect issues early, this remains an ongoing and unresolved problem for the rest of the packages.

Answer to RQ2: Cross-project collaboration within the Astropy ecosystem is driven by intentions to streamline dependency management, foster knowledge sharing, coordinate shared functionalities, manage centralized infrastructure, and synchronize release cycles. While effective collaboration is often facilitated by liaisons bridging communities, challenges such as fragmented efforts, redundant implementations, scattered issue threads, and insufficient tooling persist. Addressing these challenges through improved tools and structured community interactions can enhance the sustainability and efficiency of scientific OSS ecosystems.

6 Discussion

Our study investigates collaboration within the Astropy ecosystem, providing a unique lens to examine the convergence of interdisciplinary and open-source collaboration. This exploration highlights not only specific challenges in scientific OSS but also broader implications for collaboration research in CSCW. Below, we situate our findings within established CSCW frameworks, addressing gaps in prior work and proposing future research directions. In line with Paine et al. [130]'s assertion that CSCW is well-positioned to support scientific computing by framing software as a complex interplay of interrelated practices and artifacts (such as plots), rather than isolated code or algorithms, we also provide design recommendations for platform and tool developers.

6.1 Bridging Interdisciplinary Expertise and OSS Dynamics

The Astropy case study reveals a nuanced view of interdisciplinary collaboration in scientific OSS development, offering insights that extend beyond participant observations. While contributors possess a blend of scientific and

software engineering expertise, echoing the "spectrum of expertise" concept, this dual capability minimizes traditional communication barriers but does not eliminate deeper systemic tensions.

Prior CSCW literature on scientific collaboration (Section 2.1) identifies consistent challenges in interdisciplinary communication, often stemming from differences in professional culture and objectives [62, 89, 128, 150]. Astropy's reduced communication challenges suggest the open-source model fosters an environment where shared infrastructure and asynchronous workflows mitigate such barriers. Our observations in OSS settings reveal that locating team members who can act as intermediaries between the two disciplines is not challenging because most contributors display a range of experience in both disciplines, as indicated in the spectrum. Communication between interdisciplinary experts regarding software design is no longer a big challenge, however, the difficulty lies in consolidating different mindsets. This observation aligns with, but extends, earlier CSCW studies on team dynamics and interdisciplinary collaboration in co-located labs or academic settings [96, 111]. While communication has improved, tensions remain around task prioritization (e.g., emphasizing scientific contributions over software quality or vice versa) and perceptions of seniority, which continue to influence communication efficacy and decision-making processes. Future CSCW research could explore strategies for fostering alignment in task prioritization and decision-making across diverse OSS teams, particularly through participatory governance models or conflict-resolution mechanisms tailored for interdisciplinary collaboration.

6.2 Enhancing Inter-Project Collaboration in the Large Scientific OSS Ecosystem

The Astropy ecosystem exemplifies the complexities and opportunities inherent in inter-project collaboration within scientific OSS, showcasing both unique challenges and innovative strategies. By situating these findings within CSCW frameworks, this section contextualizes the observed practices and offers broader implications for OSS ecosystems.

Leveraging Shared Infrastructure for Ecosystem Cohesion. Astropy's reliance on centralized templates and automated tools reflects the growing need for shared infrastructure to address the complexities of inter-project collaboration. Prior studies emphasize the importance of infrastructural coherence in OSS ecosystems to facilitate coordination and reduce friction among contributors [47, 157]. The case of Astropy extends this discussion by highlighting domain-specific needs, such as dependency management and scientific workflow alignment. For example, tools like *batchpr* simplify maintenance across interconnected packages, ensuring consistency without overburdening maintainers. These practices underscore the potential for shared infrastructure to act as a unifying force in OSS ecosystems. However, the challenges of fragmented discussions and knowledge loss, as observed in Astropy, suggest that existing tools may fall short in addressing traceability and accessibility across projects. Future work in CSCW could explore the design of advanced tooling that supports ecosystem-wide traceability, leveraging machine learning to detect and connect related discussions automatically.

Boundary-Spanning Roles: The Power of Dual Membership. A recurring theme in Astropy's inter-project collaboration is the critical role of contributors with overlapping memberships across communities. These individuals act as boundary spanners, bridging knowledge and fostering communication between groups. This finding aligns with CSCW literature on the importance of boundary roles in knowledge diffusion and coordination [119]. Many contributors in the Astropy ecosystem often "*wear multiple hats*" (P10), acting as maintainers of both coordinated and affiliated packages while also serving as core contributors to the core package. For example, P5 noted, "*many of the people are both maintainers of coordinated and affiliated packages and core contributors to the core package.*" The coordinated package *astroquery* exemplifies this dynamic, as it has developed its own robust community while maintaining a strong overlap

with the core Astropy community. As P6 observed, this overlap enhances communication efficiency between the two groups.

Beyond Astropy, this concept of dual membership raises broader questions about how OSS ecosystems can intentionally cultivate and support such roles. For communities lacking overlapping contributors, as noted in the findings, the absence of boundary spanners exacerbates delays in issue resolution. These observations suggest the need for deliberate strategies to embed and sustain boundary-spanning roles, such as mentorship programs or rotational responsibilities.

Future research could explore strategies for building bridges between loosely connected communities. Prior studies suggest that rotating individuals between clusters can enhance collaboration by introducing diverse perspectives and fostering cross-pollination of ideas [143]. Examining how such practices could be implemented in OSS ecosystems may reveal effective methods for improving inter-community collaboration.

Balancing Centralization with Autonomy. Astropy's approach to inter-project collaboration illustrates the tension between centralized coordination and individual project autonomy. While centralized infrastructure simplifies workflows and ensures ecosystem-wide consistency, it can also introduce challenges related to alignment and communication during updates. This balance mirrors broader CSCW discussions on the governance of distributed groups [118], where top-down control must coexist with grassroots innovation.

Astropy's success in managing these dynamics—through tools, mailing lists, and periodic check-ins—offers a roadmap for other ecosystems. However, as the findings highlight, reliance on manual coordination and ad hoc discussions remains a significant bottleneck. Future research could investigate governance frameworks that better integrate structured coordination mechanisms with flexible, community-driven practices.

6.3 Implication for platform builders and researchers

Our research identifies challenges that hinder collaborative development in scientific OSS projects, even those deemed successful. Our findings highlight limitations in existing tools that restrict their capabilities. Next, We propose several design recommendations for platform and tool builders for automating collaboration processes and enhancing awareness and traceability.

Lowering the contribution barrier for both disciplines. Although communication between experts from different training backgrounds is less challenging than in previous years, thanks to individuals possessing dual expertise who act as bridges, relying solely on these individuals is impractical. The *Astropy* community's recent paper highlights the ongoing difficulty in finding professional software developers with robust domain-specific backgrounds [132]. Moreover, in order to contribute to a scientific OSS, the entry barrier is further elevated compared to traditional OSS, because of the need for both software engineering expertise and a scientific background for meaningful contributions. Therefore, lowering the entry barrier for contribution by listing the required expertise, including both scientific and software-related aspects is important. One idea is to *better organize "Good First Issues" (GFIs)*, which is a recommended practice to help onboard newcomers [85, 169]. Although it is already adopted by the *Astropy* ecosystem, interviewees still acknowledge the difficulty in determining suitable tasks for contribution. Future studies could explore approaches to offer more detailed guidance to "Good First Issues" that includes the essential scientific theory behind the issue and the necessary programming knowledge. Tools to break down the sub-tasks necessary to resolve the related issues would also be beneficial in emulating a step-by-step onboarding process for newcomers. Additionally, future CSCW research could explore opportunities to develop educational mechanisms that help scientists learn and engage with coding workflows more efficiently, while also enabling developers to better understand the domain knowledge underlying the codebase.

Improving the recognition of voluntary contributions. Within scientific OSS communities, contributions related to engineering are often undervalued, primarily due to the academic evaluation system. This greatly discourages individuals from participating in the community and making further contributions. To appropriately acknowledge contributors and encourage long-term commitment, it is essential to demonstrate the impact of their contributions both quantitatively and qualitatively. While existing CSCW efforts have identified factors influencing academic rewards and credits and proposed solutions, such as tools for improving software citation [68, 71, 98], our study with practitioners highlights the need for more nuanced approaches to rewarding and acknowledging contributions to OSS, such as quantifying the individual contribution impact. Future work can explore strategies for recognizing engineering-related contributions more effectively, which may involve developing improved metrics to showcase the impact of such contributions, moving beyond simply tallying downloads and citations. For example, one could examine the applicability and expand upon the Metrics proposed by CHAOSS [35] for assessing the sustainability of the scientific OSS ecosystem. This could also serve to incentivize contributors to participate more actively.

Enhanced tools are needed to efficiently summarize cross-project communication threads. As shown in our findings, people from various subcommunities collaborate for different purposes, such as sharing knowledge and creating similar features. Yet, inefficiencies remain, such as redundant efforts, lost contribution, and lack of group awareness, echoing prior studies on the collaboration within a single project and its forks [102, 174]. To enhance project management efficiency, improved tools are required for generating comprehensive summaries of threaded discussions spanning multiple projects over time. These tools should effectively organize discussions related to issues and PRs across multiple threads, rather than focusing solely on individual threads [83], potentially involving extracting valuable information and generating project documentation, aiming to enhance knowledge sharing while also ensuring that important information is not lost or scattered without appropriate management.

Such a technique can also be readily adapted to facilitate issue summarization, aiding maintainers in effectively organizing pending tasks and identifying urgent priorities. This adaptation was affirmed by interviewees when discussing their workload management:

"*... issues and PRs have been a really important way for us to keep track of everything that's going on with the project. But I think we're hitting the limits of what that kind of interface can do for collecting the relevant information...we want people to be able to know like where are the places that actually that we need help on, and so I wonder if there's a better way of consolidating that information than to have 1000 open individual issues*" (P5).

Automatically linking relevant information across projects. Moreover, there is a significant benefit in developing automated systems that can predict cross-referenced links between projects. This technological advancement not only streamlines but also enhances communication and collaboration among communities. It alleviates the traditional reliance on long-term contributors who manually connect relevant information, thereby promoting more efficient knowledge sharing and synergy across different projects and communities.

In sum, our research enhances comprehension of awareness and traceability requirements in the development of scientific OSS projects, particularly within interdisciplinary and cross-project collaborations. It identifies specific areas for future development. By applying these design implications, we aim to address collaboration challenges in scientific OSS projects and potentially extend benefits to wider open collaboration contexts, such as OSS in general, scientific collaboration [62], or collaborative hardware design [58].

7 Limitations

We employed open coding to examine the responses from the interviewees, and the discussion content mined in the *Astropy* ecosystem, which may have resulted in unintended bias. We minimized this bias by having multiple coders independently analyze the data and reach a final agreement. Taking the procedure of analysis cross-reference links as an example, two authors independently labeled 20 randomly selected links using the described steps. Subsequently, the team discussed any instances of disagreement and derived a codebook. After reaching a consensus, one author proceeded to label the remaining links and their corresponding CRGs. Whenever uncertainty arose, the team discussed the case until an agreement was reached. We also triangulated our qualitative analysis results with the version control data and code review history obtained from GitHub.

Participation bias might be introduced as the participants who took part in our studies did so voluntarily. Moreover, we used a convenience sampling method to identify interviewees to interview; this is at the risk of sampling bias. Although we mitigate this risk by interviewing contributors with different roles and responsibilities, there may be other factors shared by the interviewees that do not apply to the entire community. Caution should be exercised when making generalizations beyond the participants who were interviewed.

Our objective with this case study is not to generalize findings across the entire scientific OSS ecosystem. Instead, we aim to identify insights that may be relevant to other projects by conducting an in-depth examination of collaborative practices within a mature scientific software project comprising over 50 interoperable packages. Consequently, the characteristics of the *Astropy* ecosystem are not intended to be representative of all projects in the open-source environment. We believe that while our findings result from studying a single ecosystem, our script and codebook provide a foundational framework for conducting multiple case studies in this area, serving as a starting point to analyze and derive insights from various collaborative environments. Meanwhile, given the vast volume of information available on GitHub related to collaboration efforts, there is a compelling need to use such information to complement other research methods, such as interviews and surveys, to gain insights into communication and collaboration patterns. Our study represents a pioneering effort in investigating the dynamics of large-scale ecosystems supported by mining and analyzing the cross-referenced events in *Astropy*. At the same time, our work also reveals the need to accurately identify appropriate individuals for communication and coordination, ensuring efficient issue resolution without adding to workload burdens.

In this study, our focus was solely on cross-project collaboration within the *Astropy* ecosystem. Future research could extend these findings to the broader Python scientific community and other scientific software ecosystems to assess their applicability on a larger scale. For example, we when discussing cross-project communication, our interviewee noted challenges in communicating with projects outside the *Astropy* ecosystem. For example, P3 mentioned an instance where *Astropy* encountered an issue requiring a fix in the *Matplotlib* project, which involved multiple iterations before eventual resolution due to the lack of a dedicated maintainer for the specific sub-package at the time.

Furthermore, our study focused exclusively on a mature scientific software system. Subsequent research could investigate the long-term evolution of such ecosystems, monitoring changes and adaptations over time. By applying insights from *Astropy*, researchers could pinpoint and promote best practices to support early-stage development teams of scientific software, aiming to establish sustainable practices that foster longevity and innovation in this field.

8 Conclusion

In this study, we investigate collaboration challenges within a scientific OSS ecosystem, with a specific focus on interdisciplinary and multi-project collaboration at the ecosystem level. Our analysis centers on 50 software repositories

within this ecosystem, examining their commit histories and discussions on issues and pull requests (PRs). Additionally, we conduct interviews with code developers from the *astropy/astropy* package to address our research inquiries.

Our study reveals that the interdisciplinary team lacks a clear distinction between the roles of scientists and software developers, as both groups extensively interact with various types of code files. However, we observed tensions arising from differing mindsets—some prioritize scientific aspects while others prioritize software development—which can lead to communication inefficiencies and misunderstandings during collaboration.

Furthermore, we investigated cross-project collaboration, analyzing the reasons behind it through cross-referencing issue/PR events, supplemented by interviews with project contributors. Our findings underscore the distinctiveness of the *Astropy* ecosystem compared to previous studies focused primarily on code dependency relationships.

Moving forward, we identified challenges in collaborative development within scientific open-source software (OSS) projects and proposed strategies to overcome these obstacles, as well as suggested avenues for future research.

References

[1] 2022. GitHub-Timeline events. https://docs.github.com/en/rest/issues/timeline
[2] 2023. Astropy Affiliated Packages. https://www.astropy.org/affiliated/index.html#affiliated-packages
[3] 2023. Astropy Coordinated Packages. https://www.astropy.org/affiliated/#coordinated-packages
[4] 2023. Astropy core library. https://github.com/astropy/astropy
[5] 2023. Astropy core package maintainer. https://www.astropy.org/team.html#Core_package_general_maintainer
[6] 2023. Astropy infrastructure packages. https://www.astropy.org/affiliated/index.html#infrastructure-packages
[7] 2023. The Astropy Project. https://www.astropy.org/about.html#about-the-astropy-project
[8] 2023. astropy/astropy-project. https://shorturl.at/pxFH5
[9] 2023. Automated Pull Requests. https://github.com/astrofrog/batchpr
[10] 2023. Biopython: a set of freely available tools for biological computation written in Python by an international team of developers. https://biopython.org/
[11] 2023. CANARIE: Research Software. https://www.canarie.ca/software/
[12] 2023. Case Study: First Image of a Black HoleCase Study: First Image of a Black Hole. https://numpy.org/case-studies/blackhole-image/
[13] 2023. The Comprehensive R Archive Network. https://cran.r-project.org/
[14] 2023. CRAN package repository: Available Packages. https://cran.r-project.org/web/packages/
[15] 2023. Digital Research Alliance of Canada: Research Software. https://alliancecan.ca/en/services/research-software
[16] 2023. Eclipse Foundation: Projects. https://projects.eclipse.org/
[17] 2023. Enhanced support for citations on GitHub. https://github.blog/2021-08-19-enhanced-support-citations-github/
[18] 2023. European Commission: Horizon Europe. https://shorturl.at/kqFNS
[19] 2023. Event Horizon Telescope. https://eventhorizontelescope.org/
[20] 2023. EVERSE: European Virtual Institute for Research Software Excellence. https://everse.software/
[21] 2023. Horizon: Development of community-based approaches for ensuring and improving the quality of scientific software and code. https://shorturl.at/jtNZ7
[22] 2023. JAMES WEBB SPACE TELESCOPE: GODDARD SPACE FLIGHT CENTER. https://webb.nasa.gov/
[23] 2023. The Journal of Open Source Software. https://joss.theoj.org/
[24] 2023. Matplotlib: Visualization with Python. https://matplotlib.org/
[25] 2023. NumFOCUS: Better tools to build a better world. https://numfocus.org/
[26] 2023. NumPy: The fundamental package for scientific computing with Python. https://numpy.org/
[27] 2023. Open Source Science Initiative. https://science.nasa.gov/researchers/open-science/
[28] 2023. ReSA: Research Software Alliance. https://www.researchsoft.org/
[29] 2023. rOpenSci: R packages for the sciences. https://ropensci.org/
[30] 2023. SciPy: Fundamental algorithms for scientific computing in Python. https://scipy.org/
[31] 2023. Software Sustainability Institute. https://www.software.ac.uk/about
[32] 2023. *The Astropy Project*. www.astropy.org/
[33] 2023. Wikipedia Category: Science software. https://en.wikipedia.org/wiki/Category:Science_software
[34] 2024. The Amsterdam Declaration on Funding Research Software Sustainability. https://adore.software/
[35] 2024. CHAOSS metrics. https://chaoss.community/kb-metrics-and-metrics-models/
[36] 2024. Replication Package. https://zenodo.org/records/12631814
[37] 2024. The United States Research Software Engineer Association. https://us-rse.org/

[38] Stan Ahalt et al. 2014. Water Science Software Institute: Agile and Open Source Scientific Software Development. *Computing in Science & Engineering* 16, 3 (May 2014), 18–26.

[39] Usman Akhlaq and Muhammad Usman Yousaf. 2010. Impact of software comprehension in software maintenance and evolution.

[40] Alfred P. Sloan Foundation 2022. Better Software for Science. https://shorturl.at/swxR2.

[41] David Ameller et al. 2015. Development of service-oriented architectures using model-driven development: A mapping study. *Information and Software Technology* 62 (2015), 42–66.

[42] Elvira-Maria Arvanitou et al. 2021. Software engineering practices for scientific software development: A systematic mapping study. *Journal of Systems and Software* 172 (Feb. 2021), 110848.

[43] Wolfgang Bangerth et al. 2013. What makes computational open source software libraries successful? *Computational Science & Discovery* (Nov. 2013).

[44] Olga Baysal, Reid Holmes, and Michael W Godfrey. 2014. No issue left behind: Reducing information overload in issue tracking. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 666–677.

[45] 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.

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

[47] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. 2008. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 24–35.

[48] Christopher Bogart et al. 2016. How to break an API: cost negotiation and community values in three software ecosystems. In *Proc. Int'l Symposium Foundations of Software Engineering (FSE)*.

[49] Chris Bogart et al. 2021. When and How to Make Breaking Changes: Policies and Practices in 18 Open Source Software Ecosystems. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* 30, 4 (2021).

[50] Virginia Braun and Victoria Clarke. 2021. To saturate or not to saturate? Questioning data saturation as a useful concept for thematic analysis and sample-size rationales. *Qualitative research in sport, exercise and health* 13, 2 (2021), 201–216.

[51] BSSw 2022. Better Scientific Software (BSSw). https://bssw.io/.

[52] Thommie Burström et al. 2022. Software ecosystems now and in the future: A definition, systematic literature review, and integration into the business and digital ecosystem literature. *Transactions on Engineering Management* (2022).

[53] Amber G Candela. 2019. Exploring the function of member checking. *The qualitative report* 24, 3 (2019), 619–628.

[54] Gerardo Canfora et al. 2011. Social interactions around cross-system bug fixings: the case of FreeBSD and OpenBSD. In *Proc. Working Conf. Mining Software Repositories (MSR)*.

[55] Chaomei Chen, Yue Chen, Mark Horowitz, Haiyan Hou, Zeyuan Liu, and Donald Pellegrino. 2009. Towards an explanatory and computational theory of scientific discovery. *Journal of Informetrics* 3, 3 (July 2009), 191–209. https://doi.org/10.1016/j.joi.2009.03.004

[56] Nan-Chen Chen, Sarah Poon, Lavanya Ramakrishnan, and Cecilia R. Aragon. 2016. Considering Time in Designing Large-Scale Systems for Scientific Computing. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, San Francisco California USA, 1535–1547. https://doi.org/10.1145/2818048.2819988

[57] Zhifei Chen et al. 2022. Collaboration in software ecosystems: A study of work groups in open environment. *Inform. and Software Technology* (2022).

[58] Kathy Cheng, Shurui Zhou, and Alison Olechowski. 2024. " A Lot of Moving Parts": A Case Study of Open-Source Hardware Design Collaboration in the Thingiverse Community. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW1 (2024), 1–29.

[59] Ivan Chompalov, Joel Genuth, and Wesley Shrum. 2002. The organization of scientific collaborations. *Research Policy* 31, 5 (July 2002), 749–767. https://doi.org/10.1016/S0048-7333(01)00145-7

[60] Cecil Chua et al. 2010. Artifacts, Actors, and Interactions in the Cross-Project Coordination Practices of Open-Source Communities. *Journal of the Association for Information Systems* (Dec. 2010), 838–867.

[61] Jailton Coelho, Marco Tulio Valente, Luciana L Silva, and André Hora. 2018. Why we engage in FLOSS: Answers from core developers. In *proceedings of the 11th int. workshop on cooperative and human aspects of software engineering*. 114–121.

[62] António Correia, Shoaib Jameel, Daniel Schneider, Benjamim Fonseca, and Hugo Paredes. 2019. The effect of scientific collaboration on CSCW research: A scientometric study. In *2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 129–134.

[63] 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: IFIP WG 8.2 Working Conference, Turku, Finland, June 6-8, 2011. Proceedings*. Springer, 215–229.

[64] Kevin Crowston, Kangning Wei, Qing Li, and James Howison. 2006. Core and periphery in free/libre and open source software team communications. In *Proceedings of the 39th Annual Hawaii int. conf. on System Sciences (HICSS'06)*, Vol. 6. 118a–118a.

[65] Daniela S Cruzes et al. 2011. Recommended steps for thematic synthesis in software engineering. In *Proc. Int'l Symp. Empirical Software Engineering and Measurement (ESEM)*.

[66] 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*. 1277–1286.

[67] Yuxiao Dong, Hao Ma, Zhihong Shen, and Kuansan Wang. 2017. A Century of Science: Globalization of Scientific Collaborations, Citations, and Innovations. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Halifax NS Canada, 1437–1446. https://doi.org/10.1145/3097983.3098016

[68] Caifan Du, Johanna Cohoon, Jason Priem, Heather Piwowar, Casey Meyer, and James Howison. 2021. CiteAs: Better Software through Sociotechnical Change for Better Software Citation. In *Companion Publication of the 2021 Conference on Computer Supported Cooperative Work and Social Computing*. ACM, Virtual Event USA, 218–221. https://doi.org/10.1145/3462204.3482889

[69] Jonah Duckles et al. [n. d.]. Astropy Community Engagement. https://doi.org/10.5281/zenodo.10603048

[70] Benedikt Fecher and Sascha Friesike. 2014. *Open science: one term, five schools of thought.* Springer International Publishing.

[71] Sebastian S. Feger, Paweł W. Wozniak, Lars Lischke, and Albrecht Schmidt. 2020. 'Yes, I comply!': Motivations and Practices around Research Data Management and Reuse across Scientific Fields. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW2 (Oct. 2020), 1–26. https://doi.org/10.1145/3415212

[72] Melanie Feinberg, Will Sutherland, Sarah Beth Nelson, Mohammad Hossein Jarrahi, and Arcot Rajasekar. 2020. The New Reality of Reproducibility: The Role of Data Work in Scientific Research. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW1 (May 2020), 1–22. https://doi.org/10.1145/3392840

[73] Isabella Ferreira, Jinghui Cheng, and Bram Adams. 2021. The" shut the f** k up" phenomenon: Characterizing incivility in open source code review discussions. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–35.

[74] Bent Flyvbjerg. 2006. Five misunderstandings about case-study research. *Qualitative inquiry* 12, 2 (2006), 219–245.

[75] Jill J Francis et al. 2010. What is an adequate sample size? Operationalising data saturation for theory-based interview studies. *Psychology and health* (2010).

[76] Oscar Franco-Bedoya et al. 2017. Open source software ecosystems: A Systematic mapping. *Information and software technology* (2017).

[77] Richard B Freeman, Ina Ganguli, and Raviv Murciano-Goroff. 2014. Why and wherefore of increased scientific collaboration. In *The changing frontier: Rethinking science and innovation policy*. University of Chicago Press, 17–48.

[78] Hana Frluckaj, Huilian Sophie Qiu, Bogdan Vasilescu, and Laura Dabbish. 2024. From the Inside Out: Organizational Impact on Open-Source Communities and Women's Representation. In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering*. 36–50.

[79] R Stuart Geiger et al. 2021. The labor of maintaining and scaling free and open-source software projects. *Proc. Conf. Computer Supported Cooperative Work (CSCW)* (2021).

[80] Daniel M German et al. 2013. The evolution of the R software ecosystem. In *2013 17th European conf. on Software Maintenance and Reengineering*.

[81] Matt Germonprez, Georg J.P. Link, Kevin Lumbard, and Sean Goggins. [n. d.]. Eight Observations and 24 Research Questions About Open Source Projects: Illuminating New Realities. 2 ([n. d.]). Issue CSCW. https://doi.org/10.1145/3274326 Place: New York, NY, USA Publisher: Association for Computing Machinery.

[82] Mohammad Gharehyazie et al. 2017. Some from here, some from there: Cross-project code reuse in github. In *Proc. Working Conf. Mining Software Repositories (MSR)*. 291–301.

[83] Saskia Gilmer et al. 2023. SUMMIT: Scaffolding Open Source Software Issue Discussion through Summarization. *Proc. ACM Hum.-Comput. Interact.* Proc. Conf. Computer Supported Cooperative Work (CSCW) (2023).

[84] Saskia Gilmer, Avinash Bhat, Shuvam Shah, Kevin Cherry, Jinghui Cheng, and Jin LC Guo. 2023. SUMMIT: Scaffolding Open Source Software Issue Discussion Through Summarization. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW2 (2023), 1–27.

[85] GitHub Inc. 2022. Encouraging helpful contributions to your project with labels. https://shorturl.ac/7b5iu

[86] Morane Gruenpeter et al. 2021. Defining Research Software: a controversial discussion. https://doi.org/10.5281/zenodo.5504016

[87] Greg Guest, Arwen Bunce, and Laura Johnson. 2006. How many interviews are enough? An experiment with data saturation and variability. *Field methods* 18, 1 (2006), 59–82.

[88] Timothy C Guetterman et al. 2018. Two methodological approaches to the integration of mixed methods and case study designs: A systematic review. *American Behavioral Scientist* (2018).

[89] Noriko Hara, Paul Solomon, Seung-Lye Kim, and Diane H. Sonnenwald. 2003. An emerging view of scientific collaboration: Scientists' perspectives on collaboration and factors that impact collaboration. *J. Am. Soc. Inf. Sci.* 54, 10 (Aug. 2003), 952–965. https://doi.org/10.1002/asi.10291

[90] Charles R Harris et al. 2020. Array programming with NumPy. *Nature* (2020).

[91] Joseph Hejderup et al. 2018. Software ecosystem call graph for dependency management. In *Proc. Int'l Conf. Software Reuse (ICSR)*. 101–104.

[92] Steffen Herbold, Aynur Amirfallah, Fabian Trautsch, and Jens Grabowski. 2021. A systematic mapping study of developer social network research. *Journal of Systems and Software* 171 (2021), 110802.

[93] Michael A. Heroux et al. 2005. An overview of the Trilinos project. *ACM Trans. Math. Software* (Sept. 2005), 397–423.

[94] Christine Hine. 2006. Databases as Scientific Instruments and Their Role in the Ordering of Scientific Work. *Social Studies of Science* 36, 2 (April 2006), 269–298.

[95] André Hora, Romain Robbes, Nicolas Anquetil, Anne Etien, Stéphane Ducasse, and Marco Tulio Valente. 2015. How do developers react to api evolution? the pharo ecosystem case. In *Proc. Int'l Conf. Software Maintenance and Evolution (ICSME)*. 251–260.

[96] James Howison and James D. Herbsleb. 2011. Scientific software production: incentives and collaboration. In *Proc. Conf. Computer Supported Cooperative Work (CSCW)*. ACM Press, 513.

[97] James Howison and James D. Herbsleb. 2011. Scientific software production: incentives and collaboration. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*. ACM, Hangzhou China, 513–522. https://doi.org/10.1145/1958824.1958904

[98] James Howison and James D. Herbsleb. 2013. Incentives and integration in scientific software production. In *Proceedings of the 2013 conference on Computer supported cooperative work - CSCW '13*. ACM Press, San Antonio, Texas, USA, 459. https://doi.org/10.1145/2441776.2441828

[99] Jane Hsieh, Joselyn Kim, Laura Dabbish, and Haiyi Zhu. 2023. " Nip it in the Bud": Moderation Strategies in Open Source Software Projects and the Role of Bots. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW2 (2023), 1–29.

[100] Xing Huang, Xianghua Ding, Charlotte P. Lee, Tun Lu, and Ning Gu. 2013. Meanings and boundaries of scientific software sharing. In *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, San Antonio Texas USA, 423–434. https://doi.org/10.1145/2441776.2441825

[101] Chan Zuckerberg Initiative. 2023. Essential Open Source Software for Science. https://shorturl.at/benGI

[102] He Jiang, Yulong Li, Shikai Guo, Xiaochen Li, Tao Zhang, Hui Li, and Rong Chen. 2023. DupHunter: detecting duplicate pull requests in fork-based development. *IEEE Transactions on Software Engineering* 49, 4 (2023), 2920–2940.

[103] Marina Jirotka, Charlotte P. Lee, and Gary M. Olson. 2013. Supporting Scientific Collaboration: Methods, Tools and Concepts. *Computer Supported Cooperative Work (CSCW)* 22, 4-6 (Aug. 2013), 667–715. https://doi.org/10.1007/s10606-012-9184-0

[104] Daniel S. Katz. 2021. Towards sustainable research software. https://doi.org/10.5281/zenodo.5748175

[105] Diane Kelly. 2015. Scientific software development viewed as knowledge acquisition: Towards understanding the development of risk-averse scientific software. *Journal of Systems and Software* (Nov. 2015), 50–61.

[106] Diane Kelly, Rebecca Sanders, et al. 2008. Assessing the quality of scientific software. In *First International Workshop on Software Engineering for Computational Science and Engineering*. Citeseer.

[107] Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl. 2017. Structure and evolution of package dependency networks. In *2017 /ACM 14th int. conf. on Mining Software Repositories (MSR)*. 102–112.

[108] Amy J. Ko et al. 2011. The state of the art in end-user software engineering. *Comput. Surveys* 43, 3 (April 2011), 1–44.

[109] Anna-Lena Lamprecht et al. 2020. Towards FAIR principles for research software. *Data Science* 3, 1 (2020), 37–59.

[110] Katherine A. Lawrence. 2006. Walking the Tightrope: The Balancing Acts of a Large e-Research Project. *Proc. Conf. Computer Supported Cooperative Work (CSCW)* 15 (Aug. 2006), 385–411.

[111] Jane Li, Toni Robertson, and Christian Müller-Tomfelde. 2012. Distributed scientific group collaboration across biocontainment barriers. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, Seattle Washington USA, 1247–1256. https://doi.org/10.1145/2145204.2145392

[112] Shiyong Lu and Jia Zhang. 2009. Collaborative Scientific Workflows. In *2009 IEEE International Conference on Web Services*. IEEE, Los Angeles, CA, USA, 527–534. https://doi.org/10.1109/ICWS.2009.150

[113] Shiyong Lu and Jia Zhang. 2011. Collaborative scientific workflows supporting collaborative science. *IJBPIM* 5, 2 (2011), 185. https://doi.org/10.1504/IJBPIM.2011.040209

[114] Wanwangying Ma et al. 2017. How Do Developers Fix Cross-Project Correlated Bugs? A Case Study on the GitHub Scientific Python Ecosystem. In *Proc. Int'l Conf. Software Engineering (ICSE)*. Buenos Aires.

[115] Wanwangying Ma et al. 2020. Impact analysis of cross-project bugs on software ecosystems. In *Proc. Int'l Conf. Software Engineering (ICSE)*. 100–111.

[116] Konstantinos Manikas. 2016. Revisiting software ecosystems research: A longitudinal literature study. *Journal of Systems and Software* 117 (2016), 84–103.

[117] Konstantinos Manikas et al. 2013. Software ecosystems–A systematic literature review. *Journal of Systems and Software* 86, 5 (2013), 1294–1306.

[118] Gloria Mark. 2002. Conventions and commitments in distributed CSCW groups. *Computer Supported Cooperative Work (CSCW)* 11 (2002), 349–387.

[119] Gloria Mark and Steven Poltrock. 2003. Shaping technology across social worlds: groupware adoption in a distributed organization. In *Proceedings of the 2003 ACM International Conference on Supporting Group Work*. 284–293.

[120] Matthew B Miles and A Michael Huberman. 1994. *Qualitative data analysis: An expanded sourcebook*. sage.

[121] Reed Milewicz et al. 2019. Characterizing the Roles of Contributors in Open-Source Scientific Software Projects. In *Proc. Working Conf. Mining Software Repositories (MSR)*. 421–432.

[122] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, and Christian KaUstner. 2022. " Did you miss my comment or what?" understanding toxicity in open source discussions. In *Proceedings of the 44th International Conference on Software Engineering*. 710–722.

[123] Chris Morris and Judith Segal. 2009. Some challenges facing scientific software developers: The case of molecular biology. In *2009 Fifth int. conf. on e-Science*. 216–222.

[124] Gaia Mosconi, Qinyu Li, Dave Randall, Helena Karasti, Peter Tolmie, Jana Barutzky, Matthias Korn, and Volkmar Pipek. 2019. Three Gaps in Opening Science. *Comput Supported Coop Work* 28, 3 (June 2019), 749–789. https://doi.org/10.1007/s10606-019-09354-z

[125] Golam Mostaeen, Banani Roy, Chanchal Roy, and Kevin Schneider. 2019. Designing for Real-Time Groupware Systems to Support Complex Scientific Data Analysis. *Proc. ACM Hum.-Comput. Interact.* 3, EICS (June 2019), 1–28. https://doi.org/10.1145/3331151

[126] Luke Nguyen-Hoan, Shayne Flint, and Ramesh Sankaranarayana. 2010. A survey of scientific software development. In *Proceedings of the 2010 ACM- International Symposium on Empirical Software Engineering and Measurement - ESEM '10*. ACM Press, Bolzano-Bozen, Italy, 1. https://doi.org/10.1145/1852786.1852802

[127] Masao Ohira et al. 2005. Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. In *Proceedings of the 2005 int. workshop on Mining software repositories*. 1–5.

[128] Agnieszka Olechnicka, Adam Ploszaj, and Dorota Celińska-Janowicz. [n. d.]. The Geography of Scientific Collaboration. ([n. d.]).

[129] Drew Paine and Charlotte P. Lee. 2017. "Who Has Plots?": Contextualizing Scientific Software, Practice, and Visualizations. *Proc. Conf. Computer Supported Cooperative Work (CSCW)* (Dec. 2017), 1–21.

[130] Drew Paine and Charlotte P. Lee. 2017. "Who Has Plots?": Contextualizing Scientific Software, Practice, and Visualizations. *Proc. ACM Hum.-Comput. Interact.* 1, CSCW (Dec. 2017), 1–21. https://doi.org/10.1145/3134720

[131] Price-Whelan et al. 2013. Astropy: A Community Python Package for Astronomy. *Astronomy & Astrophysics* (Oct. 2013).

[132] Price-Whelan et al. 2022. The Astropy Project: sustaining and growing a community-oriented open-source project and the latest major release (v5. 0) of the core package. *The Astrophysical Journal* (2022).

[133] Adrian Price-Whelan et al. 2022. The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5. 0) of the Core Package. *The Astrophysical Journal* (2022).

[134] Adrian M Price-Whelan et al. 2018. The astropy project: building an open-science project and status of the v2. 0 core package. *The Astronomical Journal* 156, 3 (2018), 123.

[135] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. 2020. Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions. In *Proceedings of the ACM/ 42nd int. conf. on Software Engineering: New Ideas and Emerging Results*. 57–60.

[136] David Ribes and Thomas A Finholt. 2007. Planning infrastructure for the long-term: Learning from cases in the natural sciences. In *Proceedings of the Third International Conference on e-Social Science*. Citeseer.

[137] Jane Ritchie, Jane Lewis, and R Gilliam Elam. 2013. Selecting samples. *Qualitative research practice: A guide for social science students and researchers* 111 (2013).

[138] Romain Robbes, Mircea Lungu, and David Röthlisberger. 2012. How do developers react to API deprecation? The case of a Smalltalk ecosystem. In *Proceedings of the ACM SIGSOFT 20th int. Symposium on the Foundations of Software Engineering*. 1–11.

[139] Danielle Robinson and Joe Hand. 2019. Sustainability in Research-Driven Open Source Software. (2019).

[140] Kristian Rother, Wojciech Potrzebowski, Tomasz Puton, Magdalena Rother, Ewa Wywial, and Janusz M Bujnicki. 2012. A toolbox for developing bioinformatics software. *Briefings in bioinformatics* 13, 2 (2012), 244–257.

[141] Per Runeson et al. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* (2009).

[142] Johnny Saldaña. 2021. The coding manual for qualitative researchers. *The coding manual for qualitative researchers* (2021), 1–440.

[143] Niloufar Salehi and Michael S Bernstein. 2018. Hive: Collective design through network rotation. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–26.

[144] Rebecca Sanders and Diane Kelly. 2008. Dealing with Risk in Scientific Software Development. *Software* 25, 4 (July 2008), 21–28. https://doi.org/10.1109/MS.2008.84

[145] Judith Segal. 2004. Professional end user developers and software development knowledge. (2004).

[146] Judith Segal. 2007. Some problems of professional end user developers. In *Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*. 111–118.

[147] Judith Segal. 2008. Scientists and software engineers: a tale of two cultures. (2008), 8.

[148] Forrest Shull et al. 2007. *Guide to advanced empirical software engineering*. Springer.

[149] Jens-Henrik Soeldner. 2021. *Understanding Social Research Networking Sites*. Springer Fachmedien Wiesbaden, Wiesbaden. https://doi.org/10.1007/978-3-658-31575-7

[150] Diane H Sonnenwald. 2007. Scientific collaboration. *Annu. Rev. Inf. Sci. Technol.* 41, 1 (2007), 643–681.

[151] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. [n. d.]. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing* (New York, NY, USA, 2015) *(CSCW '15)*. Association for Computing Machinery, 1379–1392. https://doi.org/10.1145/2675133.2675215 event-place: Vancouver, BC, Canada.

[152] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F Redmiles. 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology* 59 (2015), 67–85.

[153] Katherine J Stewart and Sanjay Gosain. 2006. The impact of ideology on effectiveness in open source software development teams. *MIS quarterly* (2006), 291–314.

[154] Klaas-Jan Stol et al. 2018. The ABC of software engineering research. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* (2018).

[155] Tim Storer. 2018. Bridging the Chasm: A Survey of Software Engineering Practice in Scientific Programming. *Comput. Surveys* 50, 4 (July 2018), 1–32.

[156] Harsh Suri. 2011. Purposeful sampling in qualitative research synthesis. *Qualitative research journal* 11, 2 (2011), 63–75.

[157] MM Mahbubul Syeed and Imed Hammouda. 2013. Socio-technical congruence in oss projects: Exploring conway's law in freebsd. In *IFIP International Conference on Open Source Systems*. Springer, 109–126.

[158] Xin Tan et al. 2022. An exploratory study of deep learning supply chain. In *Proc. Int'l Conf. Software Engineering (ICSE)*. 86–98.

[159] Erik H Trainer et al. 2014. Community code engagements: summer of code & hackathons for community building in scientific software. In *Proceedings of the 18th int. conf. on Supporting Group Work*.

[160] Erik H. Trainer, Chalalai Chaihirunkarn, Arun Kalyanasundaram, and James D. Herbsleb. 2015. From Personal Tool to Community Resource: What's the Extra Work and Who Will Do It?. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, Vancouver BC Canada, 417–430. https://doi.org/10.1145/2675133.2675172

[161] Bogdan Vasilescu. 2014. Software developers are humans, too!. In *Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, Baltimore Maryland USA, 97–100. https://doi.org/10.1145/2556420.2556833

[162] Bogdan Vasilescu et al. 2015. Gender and tenure diversity in GitHub teams. In *Proceedings of the 33rd annual ACM conf. on human factors in computing systems*. 3789–3798.

[163] Lei Wang et al. 2019. Toward the health measure for open source software ecosystem via projection pursuit and real-coded accelerated genetic. (2019).

[164] David Gray Widder et al. 2019. Barriers to Reproducible Scientific Programming. In *Proc. Int'l Symp. Visual Languages and Human-Centric Computing (VLHCC)*.

[165] Erik Wittern et al. 2016. A look at the dynamics of the JavaScript package ecosystem. In *Proc. Working Conf. Mining Software Repositories (MSR)*. 351–361.

[166] Claes Wohlin et al. 2012. *Experimentation in software engineering*. Springer.

[167] Yulun Wu et al. 2023. Understanding the Threats of Upstream Vulnerabilities to Downstream Projects in the Maven Ecosystem. In *Proc. Int'l Conf. Software Engineering (ICSE)*. 1046–1058.

[168] Laerte Xavier, Aline Brito, Andre Hora, and Marco Valente. 2017. Historical and impact analysis of API breaking changes: A large-scale study. 138–147.

[169] Wenxin Xiao et al. 2022. Recommending good first issues in GitHub OSS projects. In *Proc. Int'l Conf. Software Engineering (ICSE)*.

[170] Yunwen Ye and Kouichi Kishida. 2003. Toward an understanding of the motivation of open source software developers. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 419–429.

[171] Robert K Yin. 2009. Case study research: Design and methods Sage Publications. *Thousand oaks* (2009).

[172] Jia Zhang. 2010. Co-Taverna: A Tool Supporting Collaborative Scientific Workflows. In *2010 IEEE International Conference on Services Computing*. IEEE, Miami, FL, USA, 41–48. https://doi.org/10.1109/SCC.2010.99

[173] Qian Zhang et al. 2021. Research Software Current State Assessment. https://shorturl.at/ceuGN

[174] Shurui Zhou et al. 2018. Identifying features in forks. In *Proc. Int'l Conf. Software Engineering (ICSE)*.