USER Findings: Chapters

Research Questions

- Do scientific open source projects think about their users?
- When and how do scientific open source projects prioritize usability & design?
- How do scientific open source software projects currently incorporate user-focused practices? What are the contributing factors that lead to successful adoption of these practices?
- Is language for usability and design used consistently? What do contributors understand about the words “usability” and “design,” and what word choices do they make when describing challenges and opportunities focusing on end-user experience?
- How do the resources, tools and infrastructure in use affect when a project team decides to prioritize usability, and when is that most successful?
- How do creators of open source research software think and act in relation to the users of their software? What are the heuristics that drive the development of their software?

Outline

1. **Summary Takeaways**
2. **Literature review**
3. **Research Process**
4. **Findings:**
   a. **Part 1. The State of Science and Research Open Source Software (SROSS) Projects**
   → A Look at the Ecosystem
   → Chapter: Values and Attitudes of Open Science and Open Source
   → Chapter: Contributors and Contribution
   → Chapter: Cross-Disciplinary Collaboration
   → Chapter: Governance
   → Chapter: Sustainability & Funding

   b. **Part 2. Project Operations**
   → Chapter: Onboarding
   → Chapter: Documentation
   → Chapter: Feedback and Knowing Users
   → Chapter: Usability
   → Chapter: Prioritization
   → Chapter: Accessibility

   c. **Part 3: Design**
   → Chapter: Perceptions of Design

→ Chapter: Design in Practices and workflows
→ Chapter: Design work in the academic context
→ Chapter: Designers' Experiences
→ Chapter: Challenges and Barriers

d. **Glossary**

Summary Takeaways

Read our executive summary here on our live USER website.

USER: Executive Summary

The Usable Software Ecosystem Research project (USER) is a research initiative that explores how open source scientific and research software (SROSS) teams understand, consider, and undertake usability and design opportunities in their projects. Supported by the Sloan Foundation, our research followed an iterative Human-Centered Design approach using multiple methods of inquiry, including desk research, surveys, interviews, and observation at community events and gatherings between November 2022 and June 2023.

This project illuminated countless insights, from a close look into how a diverse set of SROSS projects and their contributors consider their users, to how they approach the usability and broader design aspects of their projects, to the complex and often contradictory ways in which they understand and build SROSS tools.

Below we've crafted a summary of our key findings, which, though significant, fails to contain the true breadth and complexity captured in our complete set of findings. To understand these takeaways in context, please navigate to the linked associated chapters. You'll find after each finding there's suggested sections associated with the findings should you want to discover more about a particular finding and then see associated recommendations that we suggest implementing into your SROSS projects in order to address a challenge present in the finding.

Common language and respect

SROSS development often needs cross-disciplinary work. This can be challenging, as norms and common language need to be negotiated between different work cultures. In SROSS projects, there is often a disciplinary exchange between programmers and scientists. Add designers, and there is a third 'language' and working style. Effective cross-disciplinary collaboration ranges in strategy and process. Some projects prioritize mutual understanding and a state of continuous knowledge transfer between disciplines, while others take a more synergistic approach where each expert is able to excel in their domain without the burden of needing to know the ‘other’ discipline well. Regardless of the approach, mutual respect, communication and knowing the value of each role's contribution is vital to the success of design and usability in SROSS.

Find out more in sections: Perceptions of design, Governance, Cross-disciplinary collaboration in SROSS, Designers’ experiences working with SROSS projects, Design work in the academic context, Design Challenges and Barriers.

Openness and values

SROSS projects users claim values like openness, access and progress of science. Though ‘design’ itself is not stated as a central value, SROSS has an idea that usability, accessibility and user-centered improvements and practices enable the ethical values of understandable, reproducible and more robust scientific progress. The Open Source way of working is essential for the development and practice of better science.

Find out more in sections: Feedback and knowing your users, Usability, Values and Attitudes of Open Science and Open Source, Design Challenges and Barriers.

Openness, design and academia

Similarly to openness being an ethical development choice that SROSS builders believe makes for better science, they also believe openness challenges some of the difficult dynamics present in academia around ownership and competition for academic prowess. This doesn’t erase these challenges, but pushes against the assumed opposition of the individual vs. the collective. Academic bureaucracy and norms can also impede work, especially community- or user-driven processes. Academia also rarely recognises design and usability work (as is often the case with code too) and roles and often will name them differently or leverage students to produce design work as an alternative to design practitioners.

Find out more in sections: Values and Attitudes of Open Science and Open Source, Design work in the academic context.

Resourcing Design and Usability

Non-designer participants found prioritizing design as a key project value difficult. The participants say they believe design will help, but how to make space for design work was unknown at best or scary at worst. Congruously, design and usability work is seen as less vital than the coding of new features and the fixing of bugs (even if design and usability improvements might solve these).

Find out more in sections: Values and Attitudes of Open Science and Open Source, Design Challenges and Barriers, Prioritization, Funding and Sustainability.

Implementing Design practices

Whether projects establish design and usability practices depends on the perceived utility of design work; what non-designers define as design, usability, and accessibility; and the ease of implementing design changes. If design work is seen as time intensive, disruptive, requiring code overhaul, and/or frivolous, the implementation of design practices is unlikely. Even when design and usability are valued, many see it as optional, incidental, or aspirational. In this case, design implementation is only ‘worth it’ when it is clearly linked to the expansion of a user base. We observed this most potently in projects with diverse users. SROSS projects express that they want to improve tools and services. Dedicated design and usability work could help doing this, but SROSS projects do not often realize this. Good design and usability practices empower users, make tools easier to use, and require less immediate support from maintainers, which ultimately helps fulfill the purpose of research and scientific discovery.

Find out more in sections: Design Challenges and Barriers, Values and Attitudes of Open Science and Open Source, Designers’ experiences working with SROSS projects, Accessibility, Funding and Sustainability, Design practices and workflows.

The way Accessibility and Usability are understood

Accessibility was understood as the ‘findability’ of a project and how the project could be made useful for different scientific fields. Some described accessibility in terms that design professionals do – the practice of ensuring that tools are inclusive for users with physical or cognitive differences – but this was more often expressed by designers in SROSS. Usability similarly had differing definitions from multiple perspectives and connected into a users ‘skill level’ towards how they use/operate the technical functions of the SROSS such as whether a scientist/research can understand and use a string of code in a terminal program to run a command that executes a function in the SROSS.

Find out more in sections: Accessibility, Usability, Design Challenges and Barriers, Perceptions of design, Designers’ experiences working with SROSS projects.

Ongoing communication as means to learn about users

SROSS projects incorporate their communities as a source of feedback for improvements. Participants reported that many user-focused practices – onboarding, feedback, training tutorials and support – are often fulfilled, in large part, by user communities themselves. Onboarding in particular is a responsibility that is passed off to communities of users. For instance, the burden of teaching a tool is often on professors, researchers, or students to teach each other. Tool feedback and troubleshooting – key components of user research – happens organically in user community channels such as Slack, Discourse, community calls, etc. Projects teams that are active and responsive in user channels are participating in informal user research, whether they recognize these user research methods or not.

Find out more in sections: Onboarding, Documentation, Feedback and knowing your users.

The Designer perspective

SROSS designers reported feeling isolated – often, they are the only designer in a project, field of science, or who contribute to the project ecosystem, and they have few opportunities to advocate for the value of design in governance processes. Currently, there are not many designers working in SROSS, so there is a lack of peer support and SROSS design-related resources. Designers often need to justify the value of design to their fellow contributors and educate them about how design and usability work. Design and designers are rarely part of SROSS governance conversations. This makes it challenging for contributors unfamiliar with design to embrace or learn about design practices, because it is not seen as foundational. There are few pathways for non-code contributions to projects, and project hierarchies tend not to favor the design contributions.

Find out more in sections: Designers' experiences working with SROSS projects, Perceptions of design, Design Practices and Workflows, Design work in the academic context, Design Challenges and Barriers.

Recommendations

Below you will find the list of recommendations that the USER research team has compiled. Many of these recommendations are broad to allow for the implementation of them to be context specific and maintainable by the specific SROSS project or team that wishes to make changes. If you'd like to discuss specific ways to implement the recommendations please reach out by raising an issue in our Github repository that is focussed around discussion here.

Supporting SROSS Design Practice

Promote design and usability as a practice that cuts labor and time for maintainers.

Usability improvements can reduce the time that maintainers, developers and community contributors spend writing step-by-step user guides and answering customer support requests when users are having difficulties with the tool.

Find out more in sections: Design Challenges and Barriers, Onboarding.

Make the entry points to contribution beyond code clear and inclusive.

Clear contribution paths allow potential contributors to understand where they are needed and how they can be most useful.
Inclusivity increases contributions, because designers and other non-code contributors will see their work as valued and prioritized.

Find out more in sections: Onboarding, Designers' experiences working with SROSS projects, Contributors and Contributions.

Develop resources, community spaces, shared channels, and information repositories for SROSS design.

If OSS/SROSS designers share experiences and learn from each other, they can help shape best practices and reduce barriers for designers new to the ecosystem. Also, including designers in OSS/SROSS maintainer channels, and vice-versa, will help increase designer/developer collaboration and provide guidance to projects without a designer on the team.

Find out more in sections: Design Challenges and Barriers, Governance.

Demonstrate via examples that many design and usability improvements can be small, iterative, and compartmentalized.

SROSS project teams limited by time and resources might assume design and UX improvements will be a large undertaking. In many cases, design and usability improvements do not require overhauling the code.

Find out more in sections: Designers' experiences working with SROSS projects, Design Challenges and Barriers, Prioritization.

In contrast to the above, ensure a project has sufficient capacity to scope and support the work that the designer undertakes.

Design is a non-trivial practice and requires space, time, and resources. Design and usability improvements may require development time. Collaboration will increase efficiency and reduce bottlenecks.

Find out more in sections: Prioritization, Perceptions of Design.

Celebrate the value of design in SROSS with active acknowledgement, accolades, and references.

Open source culture celebrates code contributions regularly (merged code, closed issues). By celebrating design contributions, the community can show that design and usability is essential for project success. Designers will feel recognized, which will allow for retention and growth of the SROSS designer community.

Find out more in sections: Governance, Values and Attitudes of Open Science and Open Source, Usability, Contributors and Contributions.

Collaboration

Establish a common language around design and usability.

Different contributors will have different understandings of what 'design' and 'usability' mean. Agreeing at the outset what the design/usability needs are and how they will be tackled can help shape a common vocabulary. It helps create a baseline of knowledge sharing, mutual respect, and curiosity.

Find out more in sections: Perception of design, Usability, Cross-disciplinary collaboration in SROSS, Governance.

Spend dedicated time at the outset to get to know the project team's different disciplines and workflows.

Any given SROSS project can include maintainers from varying disciplines, perspectives, and skill sets. While each party does not need to understand their colleagues' work in expert-level depth, understanding each others' approaches will lead to more efficient work and successful collaboration.

Find out more in sections: Design Practices and Workflows, Design Challenges and Barriers, Governance, Prioritization.

Incorporate design and usability language and processes in product planning.

Outlining design and usability activities as a critical part of the plan from the outset would reduce friction later in the process. This can also help shift perceptions and support project leads and contributors prioritize UX/UI.

Find out more in sections: Cross-disciplinary collaboration in SROSS, Prioritization, Governance.

Support and include designers and usability experts in governance and standards discussions.

Giving designers a seat at the table will legitimize design/UX as a core component of SROSS success. Via these discussions, designers can help shift all SROSS practices to be more user-centered.

Find out more in sections: Governance, Designers' experiences working with SROSS projects, Design Challenges and Barriers.

Foster community-driven accessibility practices by forming a working group or community of people with impairments.

Involving people with impairments in the building, maintaining, coding and design of SROSS helps shape more effective platforms and reduces needs for changes later. Forming a working group or community for those who identify as such would enable a stronger platform for advocacy of needs.

Find out more in sections: Accessibility, Governance, Feedback and knowing your users

Implementing User Experience and Design

Participate in existing user communities as a lightweight way to better understand users.

Active users and developers will often communicate via mailing lists, community forums, and open community calls. Observing or participating in existing community channels helps maintainers understand how the user community is using the software, what their needs are, and how to reach them.

Find out more in sections: Feedback and Knowing your Users, Design Practices and Workflows.

Use UX methods to represent a wider community of users.

Direct communication in projects often represents only the most active community members' needs. Beginners, occasional users, and users who do not have time and resources to participate are often not represented.

Find out more in sections: Feedback and Knowing your Users, Onboarding, Accessibility.

Prioritize clear, accessible documentation and ensure its discoverability.

Navigable tutorials and documentation explaining features and related concepts helps users and contributors get started on their own, promotes adoption, and saves valuable time. Documentation is most effective when easy to find and applicable to a broad spectrum of users.

Find out more in sections: Documentation, Accessibility.

Incorporate user-centered design practices as part of onboarding.

Onboarding is key to project growth in terms of more users and contributors. Projects may grow beyond their initial use case, so capturing user needs as part of the onboarding process can help inform user-centered improvements.

Find out more in sections: Onboarding, Accessibility.

Create models out of projects that have adopted accessibility, design, and usability best practices.

Case studies of projects that incorporate best practices could inspire the projects that believe these activities are complicated. Open, clear, and thorough documentation allows other projects to borrow the pieces that would work for them.

Find out more in sections: Accessibility, Usability, Design Practices and Workflows, Designers' experiences working with SROSS projects, Governance, Funding and Sustainability.

Create standards for terminology and processes around design, usability, and user research that can be adapted by others.

Common processes and terminology on design, usability and user research can be included in governance documents. These could serve as templates to be loaded into a repository or copy and pasted as needed for other projects.

Find out more in sections: Design Challenges and Barriers, Values and Attitudes of Open Science and Open Source

Literature Review:

Research on Usability and Open Source Software

Introduction

Analyzing the difficulties to integrate open source development with methods to improve project's usability go back at least two decades to Nichols/Twidale (2003). At the time of writing, the question remains open and is still researched (e.g. recent contributions by Wang et.al. 2022). This overview looks at different key points of the research, structuring it in sections on:

- Involvement of experts
- Focus on a community of developers
- Modularity of work
- Communication and coordination methods
- Distance to end users
- Attempted interventions to integrate design and open source work

Usability Experts in Open Source Projects

Usability experts are rarely involved in open source projects (Raza, Caprez 2012; Andreasen 2006, Wang 2022). This is reflected in the research that tries to find ways to involve experts (Benson et.al. 2004, Bach 2009, Hedberg, Iivari 2009, Rajanen et.al 2015). Rajanen et.al. 2015 shows how usability experts are kept out of participation via the tactics of "non-response, social exclusion, and false acceptance". While there might be several systemic factors that makes it less likely for Usability experts to participate, there can also be an aversion to the idea of expert involvement per se, as documented by Andreasen 2006, who describes concerns that expert involvement is incompatible with the group-based deliberation by community members (see Coleman 2012 for a discussion on this way of making decisions).

Open Source Projects as communities of *developers*

Open Source projects are often defined as communities of developers or hackers. This can be already found in Reymond's "The Cathedral and the Bazaar" in which he suggests "Treating your users as co-developers". As Bødker describes: "the strong sense of emotional belonging that the community commands tends to preclude the possibility of seeing beyond their own motivations." Raymond also claims that "given enough eyeballs, all bugs are shallow." which Nicols refers to when saying: "The OSS approach fails for end user usability because there are 'the wrong kind of eyeballs' looking at, but failing to see, usability issues." Dunguid points out that the principle, even within software development, only focuses on fixing bugs, not on other aspects of development. Despite these limits to apply developer's methods to non-code creations, Rajanen (2015) notes that skills in programming can help to get design improvements accepted.

Modularity, Coherence, Architecture

A commonly cited aphorism by "Brook's Law" suggests that adding more developers to a delayed project will delay it more (Brooks, 1995: "Oversimplifying outrageously, we state Brooks's Law: Adding manpower to a late software project makes it later."). Open Source projects allow parallel development and seem to avoid this problem (Nicols 2002, Feller, Fritzgeralt 2001, Raymond 1997). However, this mode of collaboration demands modularity (Benkler 2002). Not all products can be worked on with such a peer-production approach, since they might not be easily modularized without sacrificing essential traits; Benkler suggests that e.g. "Novels…are likely to prove resistant to peer production"; Duguid (2006), too, shows that other modes of production might also run into problems, even if they already use a mode of peer-production (e.g. Wikipedia). Dunguid suggests that software production, particularly the fixing of bugs, might be a mode that works very well in peer production but the principles of it might not easily transfer to other media (like user interface designs). Hill and Monroy-Hernández (2013) evaluate the claim that peer production improves the quality for results empirically. They find that the artistic quality of computer games does not benefit from peer production.

Media, Communication, Coordination

To coordinate large projects, one might assume that roadmaps, design documents and other artifacts might play an important role in defining requirements of the software. However, such artifacts are not very prominent in many open-source projects (Alspaugh/Scacchi 2013). The coordination happens in forums and mailing lists (Bødker 2007, Alspaugh/Scacchi 2013) in ongoing deliberation. Additionally, the code itself is a resource for structuring collaboration (Bolici et.al. 2016). The coordination via ongoing communication and working on code is different from the common modes of working for designers, who often coordinate via representations of users and future products (Dix 2011 or Bødker 1998 for a general overview, but also Wilkie, 2010; Akrich 1995;) That many typical representations for UX work are absent in open source projects is also noted by Benson et.al. (2004), pointing out that both user profiles and representations of the work process for designers don't exist in the project they analyzed.

Elitism and distance from non-technical users

Achieving technological competence is particularly valued in the community of open source hackers (Coleman 2012). Learning from documentation and participation in the community of hackers is expected and can actually lead to limited support for newcomers, since helping them too much "will undercut their ability to prove their worth and intelligence within a group" (Coleman 2012)
The non-skilled person is the "other" that the hacker is not. Andreasen et.al. (2006) cites a participant saying: "...hackers code for fun, and sure it is more fun to add support for some protocol feature than fixing a dialog for grandma". (Note, that the imagined opposite to a fun task for the competent and often young and male hacker is supporting an old woman). The distance between the in-group and the out-group by their hacking ability is not new, and described in Levy's "Hackers" (2010) as how members of the MIT AI Lab in the mid to late 60s divided their social world: "Winners" who are good at hacking, and "Losers" who are not.

A distinction between knowledgeable insiders and outside people who do not know how a computer works is not limited to open source hacking; it might be common in programming in general. Phil Agre (1995) describes the idea of the user in the "technical mindset" in which the user is seen as a part of the technological system – but a particular unruly one. Woolgar (1990) details how people in his field-site, a tech company, divide the world into the creators and "the user" (in general). In usability tests, the creators tried to evaluate their assumptions about users while at the same time upholding them: If a test participant would unexpectedly have knowledge of technology, they would be seen as a non-typical user.

Strategies and tactics for UX and FOSS collaboration

To integrate user experience work and open source development, there are different strategies for change. A rough taxonomy could be categorizing the suggestions by what is changed (roles, technical infrastructure, politics, education) and whose practices are taken as a model to be adapted by others (Do UX designers learn to code or developers to adapt UX methods?)

Rajanen (2013) reports a successful collaboration when designers adapted to the code-focussed ways of work in an open source project. However, they thus risk losing focus on core values of design work, particularly representing non-technical users (Rajanen 2013) Rajanen (2015) sees designers in a position of low power in the context of open source projects which makes the adaption plausible despite the risk.

Reimayr (2006) describes collaboration with several open source projects, particularly describing the need to find people with influence, clearly defining project goals and establishing communication channels. This can be seen as focussing on project politics. Other interventions focus on technical/social infrastructure – Bødker et.al. (2007) describe the introduction of a mailing list and learning materials; Bach et.al.(2009) suggest changes to a (developer) collaboration platform, introducing a UX workspace in the software. Nichols/Twidale (2003) suggests a broad range of interventions: academic involvement, involving end users, discussion infrastructure, lightweight usability testing, involvement of experts and evangelism for design, but does not provide an evaluation of these methods.

Conclusion

The question of how design work can improve the usability and usefulness of open source software for non-developers remains open – while some researchers can report successful collaborations, there are strong differences between UX designers and open source developers in what a good outcome is and in the methods to achieve it. Thus, despite research on the topic going at least back to the 2000s, the relation remains conflictual. While the research I have reviewed here has been on UX work in open source software projects in general, it is likely that the described conflicts also exist in scientific open source software projects.

Open Source Software, Hacker Culture and their relation to Academia

Intro

The idea that code should be shared and collaborated on is strongly related to what is called “hacker culture,” a group of people concerned with the creative and self-directed use of computers and code. There are plenty of programmers who would not call themselves “hackers” but nevertheless participate in open source projects, sometimes with great enthusiasm. But the culture of hacking creates and maintains the values and practices that a lot of open source software is built upon. For various historical reasons, this is not named “open source culture”. This is because “open source” is a politically laden term, connected to “free software”. “Open Source” is sometimes used as equivalent to “Free Software” and sometimes used as opposed to it. Also, when looking at culture, both “Free” or “Open Source” focus on the software and the legal treatment of its code rather than the practices and values. Thus, particularly in sociological discourse, “hacker culture” is the more common label.

For our purposes it is relevant how hacker culture (and with that open source and free software) are related to academia.

Hacker culture’s value system mirroring academia

Hacker culture and Academia both use the idea of “meritocracy”: Fame and power rests with the people who contributed to the community's progress the most. Both cultures rely on sharing of information, imagine themselves as being part of a global community, and of progressing, step-by-step, to a greater goal. The “merit” that people have in their communities is tracked in publications for researchers and in code contributions for hackers; if their research or code is cited or used, it adds to their fame. Both cultures are, at the same time communitarian with their concerns for sharing, documenting and circulating knowledge, yet also very antagonistic and individualistic by harshly criticizing contributions by others with the justification that this is needed for both the progress of science and code, lest the objects of concern will be tainted with bad contributions.

Hacker culture’s origins in academic institutions

It is unsurprising that hacker culture and academia share similarities: hacker culture’s origins are at universities. Particularly MIT is relevant here, both as an institution in general but particularly its Model Railway Club and its Artificial Intelligence Laboratory. Before home computers were widespread, universities were one of the places where computers were available—and linked to each other via the internet, mainly to other large research universities like Carnegie Mellon, and government research institutions. This allowed hackers to collaborate and socialize through computers. Today, collaboration with online platforms seems normal, but in the late 1970s and 80s it sparked new practices to work and to imagine oneself as part of a larger community.

Conclusion

In our research, people were occasionally frustrated with conflicts between open development and academia. Both systems do not match perfectly. This should not detract from their strong similarities in many areas: Their cultural values, particularly the idea of belonging to a community in which individuals gain merit are similar, which is plausible since early hacker culture was often based in universities.

Limitations

Our research tried to cover a wide range of projects and people working on them. To understand what our results apply to, we want to speak to the research’s limitations.

Our research focused mainly on United States-based (or founded) projects with some European-based projects included. This has two reasons. Firstly, the organization that funded this research, the Sloan Foundation, focuses on research and academia in the US. Secondly, a lot of projects in open source research software are created at universities in the US or Europe, because these nations and states and their economies are typically able to fund such projects. As a project concerned with both academia and the production of open source code, our sample also reflects the implicit prioritization of over-represented identities and backgrounds in regards to gender, race, and class and other factors in both academia and open source software communities.

Lastly, a limitation present in this research was time and capacity to include and recruit additional projects and people in the research process beyond those that volunteered through our initial outreach efforts. When making efforts to include under-represented majority/minority identities and populations (BIPOC, LGBTQIA+ etc.) time, capacity and voluntary self-identification is always a factor. We made efforts to signal that we wanted to speak to people and projects that were under-represented but due to time and capacity could not extend our deadlines to include those not readily available to participate.

Process

Origin Story

We at Superbloom have dedicated ourselves to understanding and sharing how designers work and how design is practiced within open source teams and communities for years (see Open Source Design and our podcasts on Sustaining Open Source Design). With support from the Sloan Foundation, we had the opportunity to dig into these questions within academic and scientific communities in particular. We named this project “USER” — Usable Software Ecosystem Research – and defined it as a research initiative that explores how open source scientific and research software (S&R OSS) teams understand, consider, and undertake usability and design opportunities in their projects.

Methods

Our research followed an iterative Human-Centered Design approach using multiple methods of inquiry. Human-centered design (HCD) is a process centered around understanding people’s — specifically end users’ — needs, leveraging qualitative and quantitative research to generate actionable insights. Our primary activities were:

1. **Ecosystem analysis** & Community Observation

   We spent the first two months of this research project analyzing data and information about the open source research software ecosystem to better understand existing resources, programs, and efforts to support research software for science. In this period, we reviewed relevant literature on design and usability in OSS, and on scientific OSS in particular.

We also began compiling information about projects and players in the ecosystem and began building what became our Ecosystem Map, which enabled us to understand the overlapping web of tools, projects, people, and funders that make up the SROSS space.

While the USER research team delved into the papers, blogs, repos and written accounts of key terms and subjects, we also made social connections with key SROSS funders, projects, institutions and individuals across conferences and community events like CZI Open Science and OpenRIT. We've learned from past design research projects in OSS that important context and nuance is revealed through relationship building, so establishing these connections contributed to our exploratory research practice. Developing an understanding of and learning to participate in the culture and community of OSS, not unlike how OSS itself gets built and maintained, comes from connecting with other people who value the work. From these connections, we built relationships that fed the research activities and also allowed our design researchers and the scientists, researchers and designers we met to speak on equitable, social terms and gather initial human-centered insights.

This process helped us describe and situate ourselves within this particular landscape: what projects currently exist, what types of funding are available to projects, what types of support and resources exist, what types of needs have been identified in the ecosystem, the language used by projects, the tools used in the ecosystem, and the metrics and measurements that may be in use by projects.

From the scan we were able to understand what questions we needed to ask in our subsequent surveys and interviews. It also helped us to develop an outreach plan and partner with ecosystem stakeholder organizations to publicize our survey and gather enthusiasm for our research.

2. **Online Surveys (48 responses).**

   Our research and outreach at conferences helped to inform preliminary findings and hypotheses, creating the foundation for our published survey. We ran two versions of the survey (one longer version with both qualitative and quantitative questions, and one shorter version with only multiple-choice questions) to gather broad signals about how the community prioritizes, describes, and self-assesses on usability and design issues. We invited the project contributors we'd met in person and those we'd identified in our ecosystem research to participate, and shared in communication channels and email lists frequented by communities of interest. We received 48 total responses to our survey.

3. **Interviews (27)**

   We completed 27 hour-long interviews for this project from December 2022 through February 2023. Our interview script contained 32 questions across 5 key themes (Users, Design, Project start & Institutional affiliation, Groups of people, contributors, Values &

the Future). We used a semi-structured style, adapting the script to a given conversation with a participant. We spoke to a broad range of maintainers, developers, designers, scientists, researchers, funders, Open Source Program Offices (OSPOs) and stakeholders – all involved with creating and maintaining open source scientific & research software. The Interview Guide sought to gain further insight into:

- How norms in academic, science, and/or open source working environments affect the choices teams make around their users and different kinds of design interventions.
- How team dynamics and trust affects those choices.
- What teams would need to be interested in or able to prioritize usability and design in their work.

4. **Data compilation and synthesis**

Our six-person research team then collaborated virtually and in-person to organize, synthesize, and analyze the qualitative and quantitative data we'd collected through the surveys and interviews. We used a combination of methods to analyze our recorded interview transcripts, interview notes, and survey results, including:

- A collaborative Miro board where we practiced card sorting on a massive scale to organize key findings and quotes
- A qualitative code book in Notion that helped us to sort quotations from interview participants
- Qualitative and quantitative analysis of survey results in Google Sheets

Working in the Open

In our efforts to operate in the same or similar manner to open source software we wanted to be as 'open' with our research as possible. We created a repository on Github to house our early research findings that we had consent to publish and some of our thoughts as we progressed the research. As we found in the research, maintaining open projects is difficult and we quickly found that while we shared informational updates such as surveys, outreach wording and purpose of the research, we found it difficult to share early thoughts, findings, and reactions. This may have been because these thoughts were raw 'hunches', observations that lacked objectivity and assumptions that we spent time internally discussing and would not come to a communicable consensus on within our timelines. We had a team member that took on the 'openness' role but we found that that role was more time consuming and didn't afford time to both monitor the project's openness while participating in actively progressing the research. We chose to prioritize the progression of the research over keeping our open repository as up-to-date as we would have liked.. Given these challenges however, we believe in the power of openness and would amend our work scope to be more inclusive of the effort that it takes to do work in the open in a more engaging and authentic way.

Part 1: The State of Science and Research Open Source Software (SROSS) Projects

A look at the ecosystem: What are these projects? Who maintains them? How do they start?

The researchers and technologists we spoke with found their way to SROSS projects in diverse, but ultimately familiar ways. Many started out as PhD students, introduced to a software project through their research or by working in a lab. Others did the important work of convincing other academics who were developing tools to make them open. Participants mentioned having a deep desire to work on OSS in particular and seeing a serious need that their project could fill.

There was broad acknowledgment from participants that a large part of the appeal, interest, and passion that comes with working on OSS comes from the community of maintainers and users. One Drupal enthusiast from Brazil shared that in the Drupal community: *"the people are all using the technology and the software as an excuse to meet and share, but of course we are passionate about the technology, which all brought us together."* Along with shared interest and passion, the OSS community *"usually takes [a project] on and we get feedback or it takes a life of its own and people develop it and make it a lot better than it was at the beginning."* Impact, scientific progress, and commitment to OSS are all values we saw in the participants' responses, and are explored in the following chapter.

To see the true range of projects we spoke with, in discipline, size, and scope, take a look at our ecosystem map. If your project team would like to be added, please submit an issue here.

Disciplines represented:

- Vocal behavior
- Geophysics
- Oceanography
- Data science
- Biology
- Genetics
- Citizen science
- Ethnography

- Library science
- Astronomy
- Meteorology
- Earth systems
- Climate science
- Engineering
- Chemistry
- Metagenomics
- Geography

Types of software represented:
- Python data tools
- Jupyter Notebooks
- Visualization
- Python libraries
- Data analysis
- Data cleaning
- Data collection for citizen science
- Digital platforms
- Data archive
- PHP
- Digital imaging
- Creating access to HPC resources

Values and Attitudes of Open Science and Open Source

We asked interviewees and survey respondents about the core values of their software projects in order to understand what primary motivations exist for individual contributors and teams. While the responses varied, most participants spoke about impact, improving science, equity, accessibility, openness, or quality/stability of the software. To dig into the practical, symbolic, and ideological importance of these projects being Open Source projects, we also asked our interviewees what the effects of their projects being OS is: Why is OS important for their work?

Notably, no interviewees mentioned usability or design as central values of their projects.

While many discussed the utility of openness and the ability for users to contribute to scientific progress, these aspects of usability were not explicitly acknowledged as such. And, from our other sections on Perceptions of Design and Usability, we can infer that project leaders did not mean to imply an understanding or affinity to usability and design in these responses per se.

Contributors to SROSS are motivated by ethical and practical considerations.

OSS is an ethical choice
"If you get to put profit aside entirely and it's all about sharing knowledge, it's really hard to argue that things shouldn't be open."

"More often than not doing things in the open is the right thing to do."

Many participants explained their affinity to Open Source and open practices was based on an ethical argument: not only is practicing open source "the right thing to do" but it actively works against forces that interviewees disavowed, such as being profit-motivated. One respondent referred to OS as "a movement" with idealist purposes, highlighting that they contribute to OS projects to work towards a better and more ethical future.

OS is practical - despite recurring complications around maintenance and financing, the *use and ideology* behind OS is practical in essence.
"For an analysis tool, it makes perfect sense for it to all be open, because you're literally going to be coding it"

Beyond ethics, many participants referred to OS as an obviously practical way of working. In simple terms, it "makes the software better." Some emphasized that, more than anything, it's free, which is an inherent advantage. Two people mentioned that it enables researchers in the US to get government funding, because the US government requires the projects they support to be built with open-sourced code. As researchers, some respondents highlighted the practicality of OS in supporting research. OS projects enable research values such as open data, open access to information, and reuse of research infrastructure (like formats and metadata). Two respondents pointed out that other researchers, scientists, and academics are drawn to being part of "free knowledge movements" and that the free and open environments that OS projects cultivate are a factor in attracting new project contributors.

OS's practical value is also rooted in its adaptability. Multiple participants highlighted that OS tools give researchers and programmers the freedom to adapt the tools to their own needs – no matter how niche they are. One participant noted that this allows people to research "questions that would probably be otherwise unanswerable." Another said she is able to "go from theory to practice really quickly." Not only is that useful for expanding opportunities for research, but it also helps the tool and software itself improve over time. Participants said that the freedom to change things and address issues is central to OS's power.

Open Source improves and furthers scientific work and progress.

"One of the core tenets of science is that it needs to be reproducible. That's how we establish facts. With models that are developed based on computer code, it's 100% essential that that

code is shared openly so that other people can run it for themselves and prove that they are getting the same results."

"Openness leads to better science"

We heard repeatedly that OS is in ideological and methodological alignment with the scientific method: its practices are based upon replication, validation, challenges, and dialogue. Many respondents said that when doing code-based analysis, OS is critical for peers to be able to analyze and replicate findings. Beyond simple replication, OS enables science to progress and improve. As one interviewee said, *"the important part is really to be able to build upon each other's work, and test each other's work and critique each other's work."* Others emphasized that openness signals stronger science as it invites collaboration and challenges. Throughout a number of responses was a sense of altruism and dedication to the common cause of science, rather than a competitive spirit between researchers.

Openness actively helps to interrupt some of the more difficult dynamics and barriers within academia.

Following the spirit of optimism for the "good of science" noted by participants, we also heard that OS culture runs up against *"the culture in some of the fields where people want to keep their code to themselves, so that they're the only one that can do it."* Inherently, OS invites participation and forecloses the option to be exclusive and proprietary to get ahead.

Participants also emphasized that working in the open enables them to do "simple things" that are, for many reasons both cultural and institutional, difficult to achieve within academic circumstances. For example, storage of data and content for reuse and access over time is often tied up in ownership and rights issues within institutions. One respondent said that through open source practices:

> *"We are simply talking about storing content, and augmenting the chances that you can find it back. So the idea that if you publish an article, you should be able to find it associated with data that allows you to reuse them. "*

The same respondent went on to explain that commonly, researchers run into frustrations, like your data being in *"an institutional repository there in your old computer that is broken."* Open licenses allow researchers to access their data even if they don't work at the institution any longer.

Academic competition and privacy concerns often interrupt the OS research ideal.

“Openness is a continuum and has trade-offs associated with it.”

While nearly all participants espoused the virtues of Open Source, many identified limitations that interrupt “perfect” openness, including academic pressures and contextual needs for privacy. In certain situations, all code, content, and data cannot be made open. Participants referred to working with data related to sensitive subjects (like biomedical research) that for good reason cannot be made openly accessible. One person explained a compromise: their project requires a verification process to ensure that only authorized researchers can access sensitive data. Others work for government entities that keep some information under lock and key for security reasons.

On the other hand, some urged that the pressures on people working in academic contexts can lead to competition and secrecy at the expense of openness. One participant shared:

> “We may think that the kind of culture of open science is idealistic and better than commerce, but in reality, professors are extremely competitive, labs are extremely competitive, and everyone is trying to get their little piece of the pie.”

A lot of this scarcity mindset is framed by the need to publish, limited resources, and fears that sharing one’s tools, data, or findings will undermine the value of a potential publication with respect to the author’s career growth

Institutionalizing Open Science and Open Source at large within academia and labs requires intentional planning and cultural buy-in.

While Open Source as a practice can interrupt some of the difficulties researchers experience within academia, participants were clear that the process of codifying and popularizing Open Science practices is still an uphill battle. Many urged that this process itself needs to be carefully planned and strategically managed. Universities are not “prepped” to work openly, because *“Open source infrastructure and academic infrastructure aren’t linked or in conversation.”* Part of this broken linkage exists in logistical aspects, like data infrastructure, which one participant cited as critical to successful OS and open science. The research pipeline, from ideation to dissemination and storage, also needs to be retrofitted for openness:

> *“The element of thoughtfulness and design that goes into actually developing research is kind of lacking outside of: ‘Hey, this is an interesting problem, how do we solve it?’ There's a whole aspect of that which is: How is it applied? How do people engage with it? How do you*

publish it? How do you make it open? All those things aren't really thought about."

This need for intentionality stems from researchers doing their work in very distinct and often stressful contexts with competing incentives and specific requirements. On the most basic level, researchers are often most concerned with sustaining their work in the short term, including securing grants, publishing to achieve tenure, job stability, etc – *"why should they care about openness when they need to survive?"* one respondent said. Another brought up a resonant fear: *"people don't want to harm their CV or next grant application by sharing"* too much of their work. Openness isn't currently measureable in the "incentives economy" of academia. It is, to some, overshadowed by publishing, tenure, grants and "newness."

Recommendation
While cultural overhaul of academic incentives is certainly one way to address one of the challenges participants identified, there is also an opportunity for developing cultures of openness in a way that can give researchers the currency they require within their institutions. Where a participant identified a "growing nervousness in OSS generally about metrics and measuring" perhaps OSPOs and others at the forefront of Open strategy can seek to develop methods of measuring openness.

- OSPOs and others working on open strategies should develop methods of measuring openness.

Our recommendation/s:

- **Choose to store your design, usability and research data with institutions that are more likely to reuse it:** Work toward openness by choosing to store your research data with institutions that are more likely to reuse it, or even partnering and collaborating with people who are more likely to reuse it.

Contributors and contribution

SROSS is sustained by contributors

Contributors are commonly understood as people who work towards the mission of an open source project on a voluntary basis. A contributor's role in a project somewhat differs from that of a maintainer's. A participant who identifies as a full-time developer clarified that maintainers are accountable for the project's stability, quality and progress while contributors can contribute in their own capacity, desire and capability without being held accountable. Maintainers also often have a decision making role which contributors may not. Anyone who cares about an OS project has the opportunity to contribute to it, which is one of the biggest differentiators with commercial software development. In fact, resource constraints in OS writ large, including in academia, and research organizations necessitate contributions to ensure project sustainability. Contribution is important to projects because it is also "a way of building trust with the community, even if it's a small number of citizens who are willing or able to participate in the development and maintenance of it", said a participant who identifies themselves as a researcher.

The nature of contributions is evolving; design has yet to find a way to be recognized at large

"Contribution means people working with me on the code." The quote brings to light the kind of output a contributor adds to a project. While historically contributions were made with code or feature development, the notion of what "counts" as a contribution is evolving to include mentoring, event organization, conducting workshops, moderating forums, looking for funding, documentation, package management, supporting software distribution, design and a lot more.

People rarely brought up design as a part of contributions; in fact, only the designers we interviewed ever mentioned design as a contribution. One participant who identified as a designer said that it was their goal to work on open source but had to figure out ways without having to write HTML and CSS. Another participant who identifies themselves as an R&D engineer mentioned in the context of design that "I don't think we have the opportunity to have them contribute a whole lot."

Who are the different contributors?

Students, researchers, developers and designers are the most easily identifiable community members who contribute. "I was a student in science and software and I got to contribute to open source" recounts one developer participant. Another participant shared their story of getting students involved with usability studies.

New developers from computer science backgrounds get attracted to contributing to open source as well. Most of the contributors fall in technical, conceptual and research areas. There was an evident dichotomy whether 'users' of the software are contributors. For projects where users can and do contribute, they contribute in different ways such as helping improve the terminology, helping choose the right language, providing feedback, or helping with software distribution by sharing it with colleagues. In other cases though, users were not considered contributors when they "just used the software or resources". This dichotomy is further highlighted as one participant claimed that "many of the current contributors have been users, at some point" while another participant said that they "have a really hard time converting users to contributors".

People's passion for contribution to the community and a commitment to science brings them to open source projects while their social interactions and finding social connections makes them stick

This has been conveyed by most of the participants in one way or the other. One participant's wholesome summary is worth mentioning, in which they indicated that they contribute:

- to impact people’s lives with the software
- for the opportunity to challenge their skills, their knowledge,
- make technology accessible to people who may not have the means to pay for a license or have the freedom to change and enhance when it's necessary

There have been other individual reasons that drive people to contribute such as their emotional connection with the project; in the case of students, to build on their academic study, improve grades, and/or fulfill work study, to be associated with a widely used software, job opportunities and career advancement, overall generosity, etc.

What makes contributors stick around in the long run however, is to be a part of the community and belong in an open space. A researcher participant shared that “for all citizen science projects I've worked on, I think the biggest thing I’ve learned beyond just using the tools is that there's typically a community of users somewhere on Slack on a forum or some mode of social interaction. And reaching out to people there, I think is the social aspect is what really makes people stick.”

Projects generally have their own structures for contributions as opposed to a formal, widely accepted standard

The questions projects try to answer when setting up structures are - how to design the governance? Who gets to make decisions? Is there a decision making structure? Who all are involved? What contributions to keep or drop?

Some projects have criteria and guidelines, code of conduct, dedicated channels as a part of their contributions governance. One participant who identified as the creator and main maintainer explained:

> “We have several criteria for integrating new features in terms of number of citations. When has it been released? Is it in the scope of the projects? Is it maintainable? … [the process is that]every PR that is a technical contribution, gets reviewed, at least by two people so that we make sure that it's valuable in terms of experience, homogeneity, and document discoverability.”

While it is true that projects value contributions for their sustainability and need to be accepting of them, not all contributions necessarily make it to the software. The participant said “we sometimes say no to people’s contribution, because we think it does not fit the scope of the library. we don't want people to contribute to the project, if it's not good, to be integrated.” It does safeguard the sanctity and stability of the software. Whatever a project decides to do, it is crucial to have a structure in place. These are really important to encourage contributions and make contributors stick around. “If one hopes to grow a community around open source projects in the academic space, else it is easy for people to get burned in the process and not want to contribute anymore” highlighted a researcher participant.

Conclusion

Contributors play a crucial role in the SROSS space. Different projects have different methods and governance styles to respond to contributors and their contributions, but what's common is their part in improving projects. The contributor ecosystem is evolving and it is really dynamic today wherein anyone can support the project beyond just code. There's also a fluid nature in which people move in and out of contributor roles for a project - from being a user to a contributor and back to user, from contributor to maintainer staff and back and many such combinations.

Our recommendation/s:

- **Set contribution guidelines and governance mechanisms early on:** This helps to avoid misunderstanding and mistrust. Don't wait for it to be set till something really goes wrong. People volunteer with their limited bandwidth and hence clear guidelines will make the process effective.
- **Include contributors in the decision making process as much as possible:** It helps with the quality of contributions and their connection with the project.
- **Make the entry points to contribution clear and make it inclusive beyond code:** For example, if a designer can easily discover how they can contribute, there are better chances of building a better serving software. Likewise for other areas.
- **Encourage interactions among the contributors:** People are motivated to share and learn from other people, find social connections, and seek a sense of community.
- **Share the project/product vision with contributors:** Many are passionate about being a part of something larger.
- **Offer feedback to contributors:** Consider setting up a periodic cadence to keep bringing them back.

Cross-disciplinary collaboration in SROSS

Intro

Scientific Open Source projects are distinct in their collaborative nature between developers and scientists. This can take different forms – in some projects, scientists muscle through programming that they're not entirely familiar with and seek help from developers down the line, some scientists have developers building tools for their use, and some teams are made up of devs and scientists working together to build the tool. But as one participant said, these projects are *"interdisciplinary by necessity,"* as they require both research/science and programming expertise to achieve their goals. Our research made it clear that interdisciplinary work at such a high level is tricky, but that many projects have made strides in understanding how to collaborate effectively.

Interdisciplinary work presents what many call a language barrier that makes collaboration challenging.

- *“Scientists and engineers think differently, but they don’t think they think differently”*
- *“Designers and Scientists live in different spheres? Are designers in visuals? advertising? business? maximize clicks! metrics are extremely different to SROSS users. Scientists don’t care about animation/billboard design”…“Interdisciplinary is difficult, it’s almost like a language barrier”*
- *It's all about communication, finding a common language, because that's another barrier that I found.*

Many scientists are not trained as programmers, or are only proficient in one language, which makes project maintenance and contribution difficult in certain project environments.

Many projects acknowledged that scientists typically do not have coding skills as strong as a traditional programmer. One participant begrudged this as a systemic issue: code development is not acknowledged as an integral part of a scientific career, when it should be.

Effective cross-disciplinary collaboration ranges in strategy and process. On the one hand, some projects prioritize mutual understanding and a state of continuous knowledge transfer between disciplines, while others take a more synergistic approach where each expert is able to excel in their domain without the burden of needing to know the “other” discipline well.

“I don't have a PHD in astrophysics but I work with people who do, and my favorite hobby is asking questions.”

Some maintainers of projects were very interested in a kind of exchange-based collaboration between the developers and scientists on their team. Some approach this from an accessibility perspective, looking to lower the barrier of entry to code contributions or what code is doing in a project. One maintainer noted, *“there's a lot of people that would use our system that aren't computational scientists. So all the code that I write, I really try and make sure that it's as accessible as possible for someone who might not have coding background.”* Beyond code, this strategy comes up in process and language as well. Some participants explained that they’re cognizant to ensure they’re *“not making assumptions in the language that we use”* when explaining OS systems to scientists.

On the other hand, some developers take an approach that allows each expert to “stay in their lane,” which to them, means giving everyone the tools and space to excel in their area of expertise. Such devs try instead to *“take the code out of the way”* in order for scientists to *“not worry about the hurdle of learning programming to solve your problem.”*

Perceptions that contributors need a precise skillset at the intersection of science and programming can be a barrier to finding contributors and converting users to contributors.

Some participants stated that a fundamental maintenance issue they experience is difficulty “finding folks with the right intersection of skills to be able to contribute to the project.” Lack of the right people, they said, means that the learning curve for contribution is steep, contributors do not understand the importance of software quality, and issues are more difficult to address because there aren’t enough properly trained people to discuss them with complexity.

Some, however, identified this attitude as a sustainability issue for these projects and the field in general. They advocated for ensuring access to collaboration between disciplines, but not requiring each contributor to know everything about both coding and science, especially for projects that are more niche. One participant described that for some projects focused on a very specific problem within a specific subdomain of science, maintainers have an idea that to be a contributor, one must *“be both an expert in some type of compiled programming language, which is vanishingly rare among most life scientists, and you also need to be an expert in this very specific sub domain”* of science. In this participant’s opinion, that requirement of expertise on both the technical and scientific sides meant foreclosing the possibility of contribution from people with one skillset or the other, and they advocated instead for *“lowering the activation energy needed to engage with a project.”*

Our recommendation/s:

- **Create and maintain common languages:** Make sure design, usability, OSS and science and research are well explained and maintained.

Governance

Introduction; what is governance in open source, what unique understandings, perspectives and applications do science and research OSS and designers have on governance.

Governance is typically expressed in OSS projects as a set of rules and shared norms and responsibilities across the project. These can be enforced at various levels, from casually and socially to formally and legally. Licensing and code of conduct are parts of the broader term ‘governance’ but different OSS projects may or may not use the term ‘Governance’ when describing how the open source is built, maintained and, in certain cases, 'owned’. Science and Research OSS has a few unique values that are considered when governance is either implicitly known or explicitly stated, though smaller-scale, early stage projects tend to lean implicit and larger or more time-mature projects lean towards explicit governance. These unique values can be found better explained throughout part one of this findings document as well as in part two under prioritization. The value that comes into play most strongly for Science and Research OSS is openness that encourages and promotes reproducibility and replicability via use of the scientific method as well as openness by way of access to as many interested people as possible so that both of these might lead to better, more robust and impactful scientific and research knowledge and discoveries.

Design and designers are rarely part of conversations about the governance of Scientific and Research OSS and therefore, don't get to participate and advocate for design, designer, usability and user focused needs in governance processes.

There are many possible reasons that could explain design and designers' lack of involvement in governance. This could be due to designers often being brought in at later stages of a project, after governance discussions and decisions have been largely made. Designers not having a culture of governance in its operations and education processes that leads to not being knowledgeable about governance generally until encountered in OSS. Or design and designers being a misunderstood and less-valued part of software builds by OSS culture generally. Regardless of the combination of factors involved in design and designers being omitted from governance, the comments made by our participants about how important design methods are to the strategic direction of an SROSS was evident when they described specific situations such as wanting user validation and feedback on what should be improved in the SROSS before moving forward with a technical development.
A designer who works out of a university lab across multiple Science and Research OSS projects, including a data cleaning and transformation project, details below aspirations of how users' voices can be included consistently and explicitly through a revamped governance initiative.
"In 2023: want to revamp governance and try to have what they want to call an "ambassador council" — 10 to 20 people to represent the community that meets 1-2 times per month to give feedback from the major user groups and have the people who understand the project and can represent the community, and also start to make introductions to new communities or groups of people. still need to hammer out the details"

To illustrate the frustrations that those coordinating contributions on projects can express, a community manager for a Python based interactive data visualizations tool told us the following.
"That original design choice of [OSS project]…it's a millstone around the library's neck. Because the design choices that were made because of [OSS project], they make no sense. Right now, we're having this crazy fight about annotations and hours. And it's… the reason this function exists and has 15,000 knobs, because that's how it was implemented in [OSS project]."
In this example, the participant makes a direct reference to an interface element (annotations and hours being a UI function that has many 'knobs' or configurable buttons associated with that UI feature). We can clearly hear the confusion at how and why certain design choices have been made in the OSS project and the conversations and arguments that can linger when there

is not clear direction on design and informed user needs at a respected level in an OSS project. Including design and user perspectives and respecting those perspectives can go a long way to settling disagreements better and clarifying decisions with tangible and applicable user research.

The same community manager goes on to describe a connected problem when trying to bring design, or any ‘non-code centric’ perspectives into an OSS project space, where non-code rules and functions are seen as ‘less technical’ and often less valuable by association. This makes finding and keeping a seat at a decision making table difficult and time consuming.
“Everyone says, like, just keep all arguments technical, it's like, nothing is only technical. Right?”
This often means that designers involved in OSS are at risk of spending years trying to assert their relevance to a project and eventually exit a project. This community manager wisely identifies that problems and their solutions are rarely ‘purely’ technical in nature, which can be difficult for many coder-centric OSS projects to come to terms with.

Conclusion

In this section we focussed heavily on the aspect of governance that most impacts design, designers, usability and user voice. However, there are many broad and complex challenges with governance that were not captured here that also impact on these design aspects. What we can assert is that If design is not given a seat at the governance table, or has to fight for an extended time, under relentless questioning, many are likely to seek more inclusive and accepting spaces where they can offer their valuable contributions and be welcomed and appreciated. This also stops people who do not identify as designers from participating in design processes and practices which are often collaborative in nature.
A community manager speaks to the lack of processes in SROSS projects contributing to the problem of users to contributors conversion challenges *“If we had better processes, I think that people like our users who could be developers, but we don't have the processes in place, and policies, we don't have the processes in place. Why is it? because we have this kind of weird gatekeeping around who is what?”.* They have stumbled on a discovery around gatekeeping and what contributions are valued by the systems of governance (code) and what is not (non-code) where the suggestion of, perhaps not all users who could be contributors would want to contribute as a developer - perhaps they would want to contribute design or other non-code contribution. Here, if it was explicit in the values and governance system of this project that design and non-code contributions are welcomed then perhaps more users would become contributors.
And finally a researcher working on a citizen science science platform by and for autistic individuals states how the design of governance itself is an important piece of an open science project.
“Front-end usability. In academic and citizen science projects: governance design is really important.”

Our recommendation/s:

- **Ensure that there are modules on governance where OSS is being taught (or can be taught) in design programs:** Include what that looks like across OSS and Science and Research OSS.
- **Develop and document common ways design, usability and user research can be included in governance documents:** Creating easy to use templates that could be loaded into a repository or copy pasted into governance documents.
- **Support and include designers and usability experts on boards:** Designers sitting on governance and standards discussions and boards for OSS can advocate better for design and usability in OSS and SROSS generally.
- **Create, surface and then empower, mechanisms and processes in SROSS that value and celebrate design, usability and user input:** Making this as vibrantly accepted as code contributions are. Discovering the best ways that those performing design in OSS can be celebrated and making sure those accolades are of broad career use. e.g. A SROSS project that uses https://allcontributors.org/ makes sure to raise up design contributions as highly as code contributors.

Funding and Sustainability

For OSS within science and beyond, funding and sustainability are of continuous concern. Championing openness above profit, projects often struggle to maintain funding, have majority-volunteer maintenance teams, or have to spend a large portion of their time fundraising. Participants reported that grants are particularly challenging: institutions take percentages of grant money as high as 60%; grants are usually limited to a single idea or feature; grants are typically focused on ‘newness’ rather than long term stability, so many projects can get funding to get started but not to keep going. Despite this being a well-known issue in the OSS community, our research revealed intricacies and considerations that come up in SROSS projects in particular.

Sustainability considerations depend greatly upon project size and scope. A spectrum of projects exist in the ecosystem: On one extreme, “niche” projects can secure funding to support a specific research question, and on the other, large, foundational tools are almost too vital to the ecosystem to face project-ending circumstances. Participants warned of the difficulties and the potentials of these extremes.

Interestingly, the most “niche” projects we spoke to – ones in which only one person built and maintained a piece of software and the project served a very specific, niche purpose – did not have as much of a maintenance and sustainability burden as slightly larger projects. The “in-between” projects are trickier because, as one participant described: *“people have a particular ambition and get a grant to make something, then the money runs out and things* *struggle to continue growing or even staying at the level they were."* Understandably, projects cannot maintain their level of development if all of a sudden their maintainers go from paid to 100% volunteer. Either they settle for volunteer maintenance and hope to attract more contributors, or they have to search for more money.

Slightly bigger or perhaps more well-known or established projects also struggle with sustainability and stress about funding, but one participant urged a reframing of the problem. This person told a story of asking a maintainer who was anxious to find new funding, *"'What, what would happen if you guys just stopped?' And he said, 'Well, I think the users would probably pick up the software and would keep it going. Because there's a lot of different user groups that depend on it.'"* So to them, *"the challenge is less 'can we keep the same people working on the same software?' but rather 'can we create software that can be passed to whoever is willing to maintain it so we don't have to start again?'"* This approach speaks very much to the ethos of both Open Source and science, in which work progresses based on the contributions and work of those that came before you. This also suggests an awareness of 'succession planning' for OSS but was not explicitly talked about by our participants. This approach, of course, is easier said than done when people are relying on project funding as their primary salaries or as part of their academic work.

Large projects with the ability to be adapted to many different uses appear to have the least trouble with sustainability, both because so many types of users rely on the software, and because the breadth of use opens up possibilities for business partnerships that smaller projects don't have as much access to. Some projects of this kind attribute such flexibility to being Open Source; OS enables them to be used flexibly. With commercial partnerships, OS enables clients to build upon an open package and make it work for their needs. It is often simply faster for companies to pay the core maintainers to do their adjustments rather than figure it out themselves.
To suggest an explicit example, projects such as jupyter Notebooks and Wikipedia are 'too big to fail'. In that an existence where these project may cease to be supported in the ways in which they are currently known could be viable (in that they could feasibly cease to be financially supported in the ways they are now) but the essential service would not 'cease' to exist without a rallying of OSS community effort in order to either maintain at a reduced capacity or to gracefully sunset and grow a new alternative.

Sustainability concerns affect both how projects choose to prioritize work and their likelihood of incorporating design practices.

One respondent said that their project prioritizes issues and features based on if it is *"something that seems like it's going to increase the sustainability of the software by bringing in funds that will actually let us keep working on it."* This speaks to a kind of desperation that funding forces onto these projects. Funding also makes certain activities, like design, seem "unimportant": *"I don't feel like I've ever worked on anything that's been large enough and well enough funded that [design] was a part that we thought was important."*

Over reliance on students as maintainers of OS projects can, however, present a sustainability challenge.

For obvious reasons, if students are key maintainers of a project housed within a university, that project is likely to lose a maintainer after a couple of years. Projects are therefore wary of overreliance on student work, but also noted that they sometimes source new contributors from academic institutions because students are eager to learn and apply their skills to real-world issues.

Part 2: Project Operations

→ Chapter: Onboarding
→ Chapter: Documentation
→ Chapter: Feedback and Knowing Users
→ Chapter: Usability
→ Chapter: Prioritization
→ Chapter: Accessibility

Onboarding

Introduction; What is onboarding, why it is important and what it typically looks like broadly in OSS projects

Onboarding most commonly refers to the intentional structuring of the first interactions that a user has with a tool, service or product. These can broadly include tutorials, guided exploration of the tool, documentation, videos, in-person guidance or annotated screens in the tool that instruct the user regarding functions and locations of information or operations. Onboarding can also be unintentional and therefore, not structured in any particular way other than what was arbitrarily chosen as the 'first interactions' crafted, written or built by the calibrator of the tool.

Regardless of intentional or unintentional onboarding structure, 'onboarding' is commonly known as the first interactions and impressions a user has with a tool. Onboarding is touted as a 'make or break' part of a user's relationship to a tool and commercial tools prioritize 'good' onboarding experiences. In the commercial world a 'bad' onboarding experience could drive users to competing tools and directly lead to lack of income or market share for that tool. Onboarding in open source software broadly isn't well written about and centers historically on documentation such as 'installation/setup' and any supplementary documentation or guides on running the OSS locally for your intended user purpose e.g. processing data. The implicit 'standards' of OSS onboarding could include making disparate connected tools/packages simpler to install by combining them into one package or by relying on the user's knowledge and proficiency in asking questions towards the project maintainers or other wider community members when they run into problems while onboarding and perhaps the most colloquially common method, reading and following the documentation and through trial and error, successfully using the tool.

Science and Research OSS projects currently value onboarding methods and practices that are in and of the community as opposed to a 'designed' onboarding process that is in-situ in the tool

From our research we noted 10+ different interpretations of onboarding. These interpretations largely focussed on some standard way that OSS projects express expectations of onboarding.

These shared ways of onboarding included:

- *"An ecosystem of resources depending on what you're using the tool for"* A designer from a large cross disciplinary documentation project detailed. *"Communities develop tutorials by themselves for their specific domains. The carpentries independently developed a course. "that's how we've got our onboarding, it's really, each communities onto their own develops a little cluster, journalists got a little cluster also of training, and so on, and the work within their community"* stated a coder at a data storage and transformation project as well as describing both informal video call onboarding tutorials and chats as well as running workshops in the style of carpentries[1].
- A co-lead researcher on a citizen science project expands on the pipeline of 'onboarding' to 'contributor' more clearly by stating that long, time intensive processes of onboarding might not allow for certain users to become contributors. *"We try to avoid having the problem of having people to really onboard them in like a long winded process, because then we will just not get anyone to participate as volunteers. So we try at least to our best to design things in a way that people can onboard themselves in that there's enough how to materials and that it's easy enough to follow along the process that people can learn it quite easily by themselves."*

This connects to a common way that general OSS invites new contributors through testing and improving the experience of 'getting started' as its a process that all users and contributors will likely go through, has variations on the experiences that values difference and inclusion as well as being an accessible topic, 'being new' for contributors to expand on.

As included in our Feedback and Knowing your Users chapter, SROSS projects find community convening a multi-beneficial practice in which they can address onboarding, accessibility and usability concerns as well as gather information on how to improve onboarding, connect other with community members that will onboard others or have already created onboarding resources and subsequently build a healthy contributor community ecosystem.

[1] Carpentries teach foundational coding and data science skills to researchers worldwide.

Onboarding is considered good by default if no user ‘complaints’ are logged. This perspective only comes into question when there is a previously ‘unknown’ researcher specific usage problem with the tool is raised.

Science and Research OSS projects assume that, a 'good onboarding' experience means that users will not have problems using their tool until later in their usage pattern (as early as 1 week later) when the users will typically raise more specific problems related to their own specific research use case using a method of communication available to them (email, issue creation, community call, conversation at event etc.). This places users in an uncertain onboarding space, where they are no longer completely new to the SROSS tool and can operate it in a basic way, yet they may increasingly have specific user problems when trying to achieve the specific needs of their own research utilizing the SROSS tool.

A solo developer maintainer for an audio processing OSS tool stated:
“Went through tutorial - was easy and people are citing and using it in paper after some kind of tutorial - 1 day to a week and come back with a specific need - not that they aren’t productive.”
This version of ‘good’ for onboarding doesn’t take into account a critical component discovered in our research, when the user of an OSS tool moves from ‘getting started’ with basic operations and processes through to a research-specific and potentially unique, new way of using or manipulating the OSS tool. This understanding of ‘good onboarding’ appears to be unmeasured by the project and more of a ‘hunch’ they get from the user patterns - where our data suggests that there are multiple factors involved with users 'persevering’ through an onboarding experience depending on how critical that OSS may be to their own academic or research goals. There are 2 key data points we discovered to support this:

1. The OSS tool may be the only OSS tool open and available or ‘sanctioned’ by the research institution for a researcher's type of study.
2. The OSS may be taught within a course or module and therefore apply to a grade for the researcher.

Onboarding is difficult or impossible to standardize across science and research OSS tools or by science/research domain in a specific tool due to the unique single-user specific ways that scientists and researchers use an OSS tool for their work.

Once the initial part of onboarding has been experienced, users tend to transition into the specific use case where users either have a unique kind of research topic or process that is using the OSS to explore, or the user is a unique kind of user with specific access needs.
A co-lead researcher on a citizen science project details the specifics of designing for ‘autistic people’ which is so broad that onboarding can only be anticipated and designed up until a point where then the most appropriate response is for the user to document and contribute their own experiences and resources back to the project so that those experiences might be of use to a similar user in the future.

“We do see that depending on on the autistic person, they might need some onboarding and that it's really hard to actually design it so that it's easily usable by by every autistic person, let's say in some cases, we found very conflicting wishes of what people would like and it's very hard to account for all of those as often they are like they are diametrically opposed: Some people want it like this, and others want it like that. So somehow we need to agree collectively with the community we are working with, like what's, what's the compromise we make? And sometimes there's maybe no compromise possible.”

Another participant explains that particular projects will have specific needs that are uniquely theirs and that the ‘builders and maintainers’ of the OSS cannot know this ahead of time. *“For newcomers it requires training. Not training on how to use it. To operate the system is simple, but we don’t know the blueprint — how the system is set up for your particular project is what needs to be made”*

This speaks to an implicit tension communicated by users of science and research software raised which is that their goals and responsibility is to their research purpose, the OSS they use, while valued, are largely to achieve a certain aim or process in the larger academic pursuit of publishing, citations and stability in academia for that individual researcher. It’s understandable that these users as they onboard generally are seeking a specific, unique need for their research goals that the OSS needs to facilitate in the timeframe and effort level that that individual researcher deems as satisfactory.

A developer working on an OSS microbiome bioinformatics platform mentions here that the community focuses on understanding their specific field.

“This community of people who are working towards some more goals, which is understanding microbiomes and how they impact health in the world. Right.”

And a developer for a Python based tool speaks to wanting to help researchers focus on research and not code, coming from their own experience.
“I'm no longer a scientist, right. But I understand how their time is valuable and how hard it is for them to focus on their science. So my goal, as a former scientist, who does code now is to take the code out of the way, like, you don't have to worry about this hurdle of learning, programming and learning how to solve your problem.”

Onboarding can be the ‘responsibility’ of professors, peers and universities if that OSS tool is an essential part of an academics course or research

In certain circumstances, as mentioned earlier, onboarding was the responsibility of labs, professors or teachers where 'the OSS' is a critical part of a module, course or similar by the 'student'. The assumption being that it was 'someone else's' (teacher's) responsibility and not that of those that build and maintain the OSS.

In most circumstances, the OSS tool builders and maintainers don't know what this process of 'teaching' the OSS to students and researchers looks like, as offered by a designer working across astronomy focused OSS tools mentioned:
"I'm not sure how long it takes to learn. I know some universities teach (our OSS tool) some of the search form processes as part of a lab or part of research."

Some other astronomy OSS tools and spoken language OSS tool builder and contributors mentioned 'state of fact' how certain researchers come to learn and be onboarded onto OSS projects:
"'Compulsory learning' aka part of your graded studies" and *"Professors that teach it in a module"*

What this largely leads to is a deferred responsibility for the user's onboarding being placed on people that are likely to not have as informed understanding of the OSS. However it does support the perspective of 'users teaching users' in that the teachers/professors are users as well as their students. This reinforced the community aspect of onboarding and learning.

Depending on the nature/function of the OSS tool, certain proficiencies are expected from users in order to onboard on to the tool.

The requirement for upfront skills and knowledge is most critical to users when we consider the onboarding processes. Most Science and Research open source software have some barriers to entry, from the highest barriers being using a command line interface and knowledge of one or multiple coding languages and how to interoperate them, to the specific terminology used within certain science and research domains through to a proficiency with commonly used companion tools (e.g. Jupyter notebooks), services (e.g. Github or Gitlab), common open source processes or practices (e.g. read the docs, install packages, access our discourse forum etc.) or typical graphical user interfaces.

One developer of a Ethnography platform details how users *"Need to know some social science terminology."* in order to make the best use of the tool. It's not unrealistic to think that people would need to know about the specific scientific or research terminology associated with a domain in order to use an OSS tool built for intended use by that domain. However, when we explore the other comments from participants about other expectations of users we begin to find skills and knowledge that is dependent on coding knowledge essential to the operations of the OSS tool or specific OSS tools with their own barriers to entry, such as this participant who designs for astronomy OSS speaking about how the Jupyter notebooks tools is needed to run their OSS tool *"As far as 'what do I start first' - need to run a Jupyter notebook to guide into tutorial".*

We also find that there are data processing related expectations that are not included within an onboarding process. Here a developer of a data analysis and visualization OSS speaks about the users needing to understand what they want their visualizations to be expressed as in order to find the order of operations in the tool to achieve that.

“You need some knowledge about spatial data and how you look at that …and then …what your goal visualization is, I think [OSS tool] lets you explore some very basic things... I think there's not a huge amount of knowledge that you have to have going in, but you definitely have to be willing to explore and poke around and try a bunch of things and, you know, stuffs not going to work necessarily the first time.”

When we see ‘designed’ or ‘user centered’ onboarding experiences, these are best guesses and appear to struggle to understand how to build an onboarding experience that takes into account multiple complex and faceted entry, exit and reentry points to the use of the Science and Research OSS

In our data, we understood that people working on Science and Research OSS were largely uninformed of onboarding as an 'intentionally designed process for users to get started in said tool'. There were a few examples of an intentionally designed onboarding experience. A community manager at an OSS visualization python library speaks about how an clickable ‘ideal’ quickstart process of using the tool was implemented yet users seem to gravitate towards the community led and based tools.
“So the optimal onboarding is we just revamped the website to have a nice like getting started, click through so they could click through to getting started. And they can see how we draw the first plot in the API we prefer and then we can go next, like the plot type. So the quickstart guide, you know, that's the path we'd like people to take. But like I said, I don't think anyone actually takes it on, StackOverflow seems to be how a lot of people onboard, just jumping from one StackOverflow to another.”

Absent from this account is the information detailed in above sections about how researchers and scientists often have specific research related purposes they want to achieve as quickly as possible in the OSS tools they use which is perhaps why these users are looking towards the community communication spaces to find similar fields or purposes to their own.

Efforts to improve onboarding through methods that are prevalent in OSS and research like tutorials and documentation appear successful initially, in so much that they exist and are accessed by some users but the projects appear to struggle to truly know how useful they are to users. This designer working on a large, multipurpose research documentation ecosystem and python packages speaks to the time and effort spent on documentation and the simplification of the user experience yet lack of measurement of whether these efforts improve onboarding for users remains a mystery.
“We spent a lot of time on documentation. And I spent a lot of time simplifying the UX. So to make the onboarding easier, but I don't actually know how much easier it is.”

On first glance taking a descriptive approach like feature lists can seem to cover the ‘essential’ information a user might need to understand a tool. Here we have a statement from a developer

working on a data transformation and augmentation OSS tool, where their documentation is ‘descriptive’ of their own projects' perspectives of the features.
“Our documentation is really ‘features’ documentation, we don't have onboarding tutorials or anything like that.”. This misses the ways that users might describe the features or how they possibly use them to achieve certain objectives. Which is largely what onboarding hopes to do for users, introduce users to the possibilities.

And finally, this developer working on an OSS microbiome bioinformatics platform speaks to the difficulty users would likely have if introductory tutorials did not exist. Suggesting that users may get stuck even before a unique research use case. *“We have introductory tutorials, and they would have a pretty hard time finding their way around without those.”*

Onboarding becomes a critical part of science and research OSS if their focus becomes inviting a broader scientific community to the usage of their tools as the OSS project matures

We found that the origins of most Scientific and Research OSS starts out as a component or the focus of an individual, group or lab's research. There are other ways SROSS starts such as part of an organization or company that support functions important to publishing research, such as data visualization but typically, OSS is made to a specific research purpose and that typically means small numbers of people in the project at the initialization.
As a project grows and gains users is when we see the people who build and maintain that OSS become more concerned with an ‘onboarding process’. Now there are more than the amount of people a single person can feasibly personally onboard and this needs to become systematized to an extent which requires understanding user motivations for use of the OSS that may or may not match their own as the builder of the OSS.
A community manager for an organization that funds and supports multiple OSS for science details when it becomes difficult to onboard new users.
“Projects who really want to apply best practices for something like interoperability or things like that, the inherent effect of that is that your code base becomes much more complex, which then makes it more difficult to onboard new folks.”
Here we discover a critical point in science and research OSS’s life cycle in when there is a very tangible reason to think about users, usability and the experience of using their tool beyond their own personal experience and what part of the tool (onboarding) it directly relates to.

Conclusion

In our data, we understood that people working on SROSS were largely uninformed of the definition of onboarding as an 'intentionally designed process for users to get started'. Onboarding was the responsibility of labs, professors or teachers where 'the OSS' is a critical part of a module, course or similar by the 'student'. The assumption being that it was 'someone else's' (teacher's) responsibility and not that of those that build the OSS. The other types of people that are responsible for onboarding/training mentioned were the 'community' that supports each other or the 'resources' like documentation, tutorials or other content that people have created to 'help others get started'.

The uncertainty of what constitutes onboarding tools is raised across multiple comments and summarized with this comment by a designer who works across multiple python based scientific and research OSS *“We have a YouTube channel with some videos. I don't know if I totally consider those onboarding tools.”*

Onboarding is thought of as important when paired critically with a tool’s need for new user adoption, citation or popularity.
There’s also an unclear understanding of what users ‘really do’ when first interacting with their own particular science and research OSS tools which seems informed by the perspectives and experiences of the people we spoke to and the other that are configuring and building the OSS tools and how much they lean towards an OSS interpretation of the users of their OSS tool. Some onboarding behaviors are contradictory, such as this comment from a community manager for a data visualization OSS tool, about starting by hacking and not reading the documentation.
“Nobody reads the manual. Everybody starts by hacking, and programming first, I try to do the thing. And then when they realize they can't do the thing, they may be going look, the docs, but then Well, it's just Google search, like, you know, search advice type of answer. So I think, I think a lot of this is just like a general reflection of how folks learn to code.”
Which may be true for more coder, developer aligned users but isn’t the universal experience of users of science and research OSS where they are either ‘taught’ by other academics, by the documentation or the wider community.

Our recommendation/s:

- **Understanding what onboarding efforts exist across an existing community for the OSS:** This can help a Science and Research OSS project team to focus their user centered practice on gathering those examples from the community and collecting them in a space that is user centered.
- **Exploration of any software tool is somewhat expected of users as they onboard into it:** A ‘designed’ onboarding process might understand that many users may come in without an idea of how to achieve a particular visualization or not know what kind of visualization is best for their data and guide them through that discovery process. It seems, though Science and Research OSS expects this prior knowledge of their users or that they discover their answer via the community or documentation.
- **Onboarding is a complicated process that includes a myriad or different possible uses beyond the initial operations:** Supporting Science and Research OSS projects to stay external user-centered as they grow beyond the initial builders of the tool is a critical point in the adoption of ‘good’ design processes for onboarding which then lead to a key success metric across most science and research OSS which is more diverse users with diverse scientific focuses using and contributing to their tool.

Documentation

Introduction

Documentation in SROSS serves many purposes such as helping users navigate through the software and discover features, it helps resolve installation issues and dependencies, it enables a better way to package the software, it also helps make the code easier to access by other developers when the code is documented well. Documentation takes various forms, from a simple read-me to tutorials, workshops, case studies, examples, training, videos etc… all serving the purposes mentioned above. Software teams believe that documentation enables their users to identify and understand what the software offers while evaluating choices. One respondent claimed that “it is a way into the future in a way that other people can continue if docs are good”. Even though most of the softwares agree that it “promotes stability in project usability - allowing users to access workflows smoothly”, not all softwares are equal when it comes to managing and strategizing documentation.

State of documentation in SROSS

There’s an entire range of documentation from pretty exhaustive user guides and reference manuals to being incomplete, outdated, having sparsely populated information which one participant called “a train wreck”. Some projects tend to describe all the concepts around their software in their documents but, critically, not how to use it. Some have their features documented but do not have anything intentionally constructed for user onboarding. One project was surprised to see people’s personal accounts of using and getting started with the software when searching for their documentation. Most of the projects have their documentation in a state of continuous work-in-progress and never up-to-date. Aspects of documents such as access and discoverability are also identified as work in progress. One of the maintainer participants mentioned that their “documents, books and guides on usage of the tool is part of the improvement process”. There were also some projects that did not know the state of their project's documentation which, given the strong connection projects made between documentation, onboarding, and usability, made us wonder about how else those projects might incorporate user-focused practices.

Response to documentation

Different projects respond differently to the need for documentation. Projects rely on documentation as a reliable plug to fill in the usability gaps, depending on the level of usability of their softwares. Documentation comes to the rescue when it’s a lot harder to get something working on the UI, the usability is still clunky, and there’s no other solution in sight. Maintainers prioritize documentation when there’s a lot of questions and conversations from the users which need to be addressed, or when there is a deeper realization that the software is confusing/complex and users don’t understand how something works. It again takes a lower priority in the face of a lot of feedback and features to be worked on, or when the projects have a sense that the users will be able to find their way around their software.

User-centered approach to documentation

There isn’t a clear user-centered approach to strategizing documentation or making them accessible and discoverable. Documentation was often brought up when we talked about user and usability related topics, suggesting that projects might view a good documentation as means to make software understandable to new users (in contrast to many designers, who focus on "user-intuitive" interfaces). The most satisfying approach to documentation came from one of the projects that identified gaps in documentation when a few users, more than 1 and less than several, misunderstand how to use the software. Documentation to them, fills a usability purpose for the SROSS, but a designers perspective would suggest that good documentation is only one aspect to making a software too usable. As one project maintainer states "docs can be a filler in the usability of a platform when there’s no clear way to present some information or to completely remove the kinks in the software".
They also have introductory tutorials that are geared towards novice users and some documents that are geared towards experienced users. The most user centric approach we encountered was to use forum activities to keep the documents updated. Lack of user-centered documentation also leads to major problems in discoverability. “We document a lot but users can’t find it [easily] which is related to search and information architecture” said one of the participants who identified as a community manager.

Conclusion

It is clear from above that documentation does not take a priority parallel or next to code. Much less are its usability and discoverability a priority. Could this be owed to not defining the responsibility? Could this be a time, funding, or human behavior issue? One hypothesis explaining the lower priority of documentation is that open source (research) software projects organize around collaborating on code with other programmers. This means that non-code contributions directed to non-programmers, are seen as secondary, less motivating and less likely to garner merit in the project.
Successful projects report that documentation for them isn’t an afterthought. It is built into the development process, as and when software gets updated. Projects agree that documentation is a fall back mechanism for usability issues; it can reduce the entry barrier for users and increase software adoption. It is only justified to adopt practices that take documentation hand-in-hand with their softwares. Perhaps some projects will move out of their consistent state of WIP and finally feel caught up.

Our recommendation/s:

- **Prioritize documentation as much as building features:** It improves software adoption and usability. Tutorials and documentations explaining features and related concepts help the community get started on the platform on their own. It saves teams their valuable time which could be better spent on more valuable areas than supporting people individually.
- **Adopt practices that enable projects to manage documentation:** Without it becoming an added burden on the team.
- **Make accessibility and discoverability of documents important:** Discovery is important as writing documentation itself and ensuring these documents are useful to as broad a spectrum of users as possible.
- **Projects can benefit from making individuals or a role responsible for documentation:** This helps in cases where the project relies on contributor support.

Feedback and Knowing your Users

Introduction

Projects learn about users' needs via online and in-person venues where both creators and developers gather already, such as forums, issue trackers, and conferences. Rarely, there were mentions of formal methods like usability testing or surveys which are also used by user experience professionals. This way of learning about users is direct and easy to integrate, but might also lead to only knowing about specific parts of user communities.

User Experience methods are rarely used.

Methods that user experience professionals use are often derived from psychology or anthropology. With these methods, professionals derive results like a report, slide decks and archetypal representations of situations (use-cases and scenarios) or users (personas, target groups). If a project follows a formal process for knowing users, these results are passed on from specialized user researchers to designers and managers. Even for professional designers, these formal processes are an idealization: Practice is messier. However, user experience design professionals know how they would like their process to look like. They also emphasize the importance of a professional distance from users: "You [the designer/researcher/dev] are not the user" is a slogan.

In our research, some projects used such formal methods, however, they were in a clear minority. The two methods that are used to get to know users are surveys and usability tests. Seven of our participants mentioned using surveys to learn about their users. How the surveys were seen was varied: "we've tried running surveys, but I think they haven't, like massively worked…” Two participants regularly conducted surveys - one of them likening them to a census - but neither stated what decisions came out of the results.

Usability tests were the other formal method that was mentioned. In a usability test, people are asked to do tasks.The person leading the test makes notes or records their actions, particularly when people get stuck or experience other problems. The test is done with several participants to see if problems repeat. Three people mentioned doing usability tests; one of them talked in depth about tests and seemed to strive to have a process that would be seen as “good” by user experience professionals. However, even that person mentioned that many tests are makeshift and use easily available testers like students, a practice UX professionals call “Guerilla Testing,” methods that are also used by professionals, but are not seen as the ideal way to test software.

Developers know “the community” directly, not through research and abstractions.

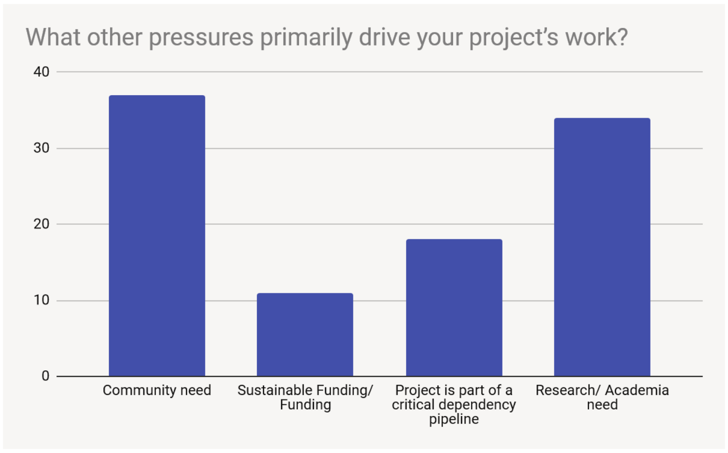
To learn about their users, projects rarely use methods of UX professionals, but other means that do not generate abstract representations but allow to relate to users in a direct way by communication: Calls, training and conferences as well as web forums, email, issue trackers and online chats.

All these communication methods are also used by software developers for writing software and participating in internet communities.They are thus not means of communicating with the users exclusively. In these online communities, there are different roles and capabilities: Users who are co-developers are rare, but often there are engaged users that know the software well. They try to figure out how to use the software, and if they are unable to do so due to limitations of the software, they voice their needs to the developers via different channels.

There is a strong focus on what people on these channels say they need. As one interview participant said:

> “[we] see ourselves essentially having like a set of stakeholders, as opposed to necessarily [a] set of users. So we have some users that are people that just want to do something, right, they want to do some science, they want to do some research and, and the system, the software that we build may enable them to do that.”

The “community need” was the strongest pressure on software development in our survey, even slightly larger than the obvious “research need.”



Communication channels are structured around the activities of developers

Of the different communication channels used, the closest to the software development are issue trackers. An issue tracker is an online platform, where problems can be described, listed and assigned to developers to work on a bug or feature. Since they are a vital part of coordinating software development, they are integrated in developer-focussed platforms like github. Software users might create issues directly. Sometimes, project members also translate mails or forum posts into an issue on their platform and thus make it more likely to influence the software itself. That this translation happens also points to the fact that writing issues is a skill that people might need to learn, as one participant pointed out: Not all ways to write about a problem lead to a “good” issue.

Other means of communicating are less development-centric. E-Mail, forum software like discourse and chat groups are common tools of web-based communities in general.Users post problems there or help other users of the software solve their problems. Some of the tools used have been around a long time (like mailing lists) and others are newer (like discourse or slack.) These tools are probably more accessible to users without socialization in open source software communities.

Working in web-based communities does not exclude face-to-face communication. Synchronous communication opportunities include :

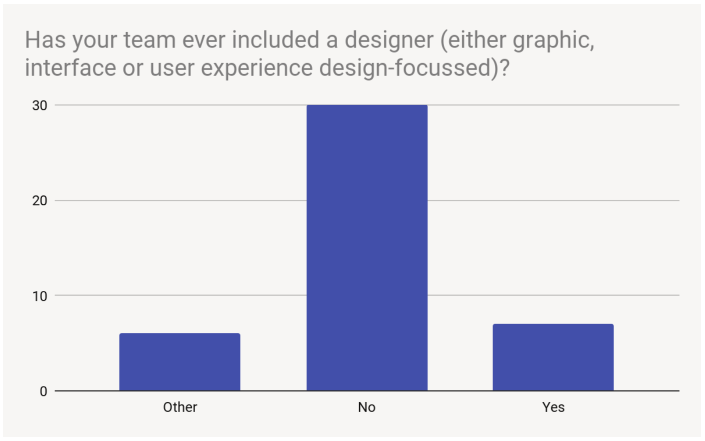
- Software training allows users to meet project teams in person.
- Conferences help project members learn how people use the software in their research and to socialize.
- Regular calls allow for remote but synchronous communication with users.

This means that people in the peripheral community of users will usually not be heard. For the users at the center of the software development community, these modes of dialogue are natural. For peripheral or less-technical users, these modes might seem hard to understand or inconvenient – if they know of them at all.

Knowing users directly: Quantitative evidence

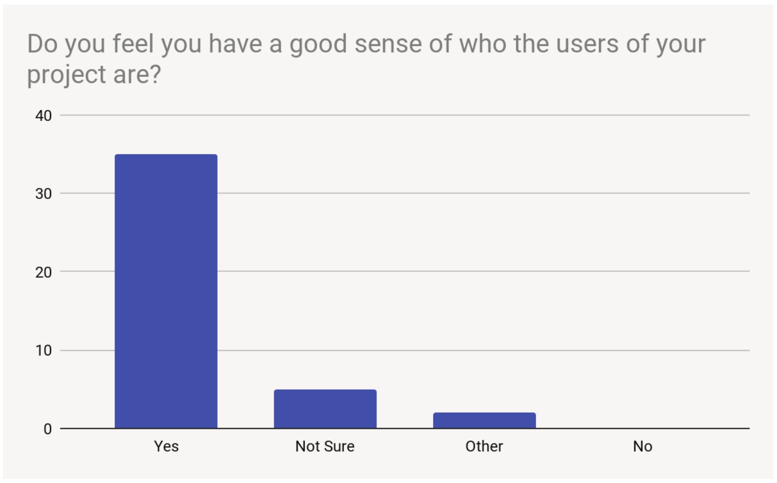
The pattern of knowing users but not using formal methods or designating separate roles for design also shows in our quantitative data:.

Only few projects mentioned thew worked with UX/UI designers:



This matches our observation that most projects did not use methods of UX professionals to get to know their users. “Knowing users well” for UX professionals usually implies a process of research: conducting data via interviews, observations, and surveys. They synthesize this data and create formal abstractions like “personas”, “scenarios” or “user journeys”, genres of documents that summarize what they learned about users. None of our projects had such research and representations. If at all, they used low-resource methods of testing existing parts of the software.

However, despite usually not having a single project member tasked with learning about users, most projects had the impression they know their users well. Not a single participant in our survey answered “No” to the question: “Do you feel you have a good sense of who the users of your project are?”



Conclusion

The idea of what it means to “know users” is very different between UX professionals and Open Source research software creators: Getting to know the user via tools that were used anyway as part of software development and online communities was far wider spread than the methods of UX professionals.

Knowing the user by means of direct communication and as part of the community of users has several advantages over the abstracted methods of UX professionals.The typical OS research software interactions with users:

- are ongoing,
- not bound to a specific functional role,
- does not depend on formal abstractions

However, the direct way of knowing users also has its shortcomings. Most importantly, these methods only allow for knowing users that are close enough to the project, have the time to be active in forums, on github, or even go to a conference. Being part of “the community”, the users visible to developers of open source research software, takes time and resources. Thus, the methods to know users might have a bias towards the most active and privileged users, whereas beginners or people with little time or money at their hands might never be known to the people creating the software.

Our recommendation/s:

- **Expect existing, direct ways to know users**: It is likely that developers in a project know their very active users well despite not using formal user research methods but direct communication.

- **Participate in existing community spaces**: Active users and developers will often communicate via mailing lists, community forums and open community calls. Participate to understand how the community works.
- **Use UX methods to represent a wider community of users**: Direct methods of communication often only represent the most active community members. UX methods can help to understand beginners, occasional users and users who do not have time and resources to participate.

Usability

Introduction; what does usability mean to people contributing to, building and designing for Science and Research OSS and what does it mean to designers outside of the science and research field.

Usability, like Accessibility discussed in a different section (accessibility), is not an unknown term in the Science and Research OSS space, yet, like Accessibility, it has many different interpretations, implications and applications depending on the perspective and motivations of the individual approaching the topic of Usability.
Below, a designer who works across many Python related SROSS tools describes where they sense the use of the term Usability is coming from the term being more pervasive in the general vocabulary of software and OSS but that the practical application of Usability falls short in their opinion.
“Usability is not necessarily an uncommon word, but something I feel like I see a lot is they kind of are picking up vocabulary, whether that's from me, whether that's from it being more present on tech spaces in the internet now, and they're reading blogs… it's very rare that I feel like it's being used as accurately as I would like it”
From our exploration and conversations we largely found that Usability was described by non-designers in the Science and Research OSS space as, discoverability of features and data (which we explore in some detail in the Onboarding section of this document), terminology and it’s appropriate use, making features and functions ‘easy’ (yet they struggle to describe what easy means contextually), and, connected to accessibility, to ensure that everyone and anyone regardless of background and prior knowledge have the opportunity to ‘access’ and ‘use’ the OSS.

Important to note in this introduction to the topic of Usability in Science and Research OSS, is that the wider infrastructure that supports OSS and Science and Research does not invest in usability in the ways in which we may come to talk about in this section. Github, Gitlab, secure University data storage systems and various other softwares and applications are usability challenged in their own ways. Some of these organizations invest in usability and design more broadly, but many people, in this research study and beyond, describe the difficulties in learning and understanding the ecosystems that the Science and Research OSS tools exist in.
“Tech tools themselves have not thought about how to make things usable and friendly. Makes it hard for, eg, undergrads who start thinking about problems and get intimidated by Git. Github is scary - buttons, menus and options…too many, I'm an outsider and it isn’t for me.”

Here, a Data research Scientist at a lab that incubates numerous open science and research tools speaks to the complexity of the software and tool ecosystem.

A challenge we heard from designers is that they can lack the knowledge of both the larger OSS ecosystems that SROSS exists in as well as the scientific and research purposes therein the OSS. With dedicated learning time and collaborative support the designers were able to understand enough about OSS and the Science/Research to design effectively, but this lack of knowledge and understanding goes both ways.
“I don't think that many of the partners that we've worked with knew what usability testing was” a designer who works in a university based lab describes the OSS team’s lack of knowledge about usability testing processes as they began working together. From these insights we can make the reasonable assumption that there are knowledge gaps from the OSS, Design/Usability and Science/Research perspectives all around to better communicate into the future.

Usability is important to Science and Research OSS when they consider themselves ‘no longer niche’ (single digit users). When there are more diverse users, or when another stakeholder has expressed interest, is when Usability becomes critical.

At the point where users are using a tool in the double digits is a stage that most designers would describe as a late stage for usability and UX improvements. This isn’t uncommon in resource constrained proprietary and commercial software, to bring in usability concerns post functioning software. But this does contribute to an aspect often raised throughout the research by our participants that design, usability and accessibility is viewed as laborious, time consuming and difficult to implement. Not surprising given the opportunity to perform usability and ux improvements would have been during the pre-build process.

A scientist and researcher building an OSS citizen science platform describes below the lifecycle of SROSS starting out small and niche and that that is fine as long as it stays as OSS that doesn’t need to serve a purpose beyond a limited number of research papers.
“Usability doesn’t have to be important, and that’s fine: “because I feel like a lot of academic open source software really starts out in such a small niche. And it's for supporting a single research paper or a series of research papers with no expectation of anyone to really grow this into a project. And I just want to like we be clear that I think that's fine. I think it's a different kind of open source.”
Many OSS tools will remain in the single digit user bases and there are no obvious ways of knowing if any given OSS project will become a highly sought after tool beyond logical assumptions such as if tools are currently scarce for that need and if the tool has good resource investment.

Science and Research OSS tool teams, like most OSS tools, have an 'ideal' user type and level of proficiency. Most OSS build and maintain for these users or people to become these kinds of users.

A legacy that seems like it's been passed on from OSS to Scientific and Research OSS is that there is an 'ideal' user. A user that is able to understand all appropriate key terms, functions, commands, particular coding languages or know the way to self-serve that knowledge through a combination of reading documentation (if available), searching out related solutions to problems or learning the processes and skills needed to self-troubleshoot. These are the users of OSS that often reflect those that also build, maintain and configure it. The user best described as 'myself' by developers of OSS. This perspective is not as much an attitude that the people we spoke to genuinely have about users but more this legacy 'ideal' that has cut through OSS culture by way of directing less skilled and proficient users to 'read the manual'.
These users do exist, as described below by a lead developer working on a Python package for audio data, there are a good number of people that can get most of the way when using SROSS in the same or similar ways that the core developers would.
"GUI - long story short, there's a bunch of people that can pick up Python and make a thing that gets 50 to 75% of the way there, but it still doesn't really give you that sort of, you know, like usability or accessibility, that would, that would really be like, I think a game changer for the community."
This lead developer, while recognizing that these people exist and there is this pervasive sense through OSS to 'self serve' until you can use OSS proficiently, that a GUI (graphical interface) would be a benefit to their project in allowing those that cannot, for whatever reason, self-serve the knowledge needed to operate the OSS like that of the core developers.

A community manager for the various OSS projects that a funder supports and maintains describes below the reliance on norms and knowledge in a community and how this really only suffices until a point where someone gets stuck.
"In a lot of cases, people are reliant on their knowledge of set of norms in a given community, with a community either being something like whatever scientific Python or the Bioconductor community in R or something, and having a standard set of norms that are kind of like missing stairs, where if you don't understand this one weird thing about how the software is interpreting your data or how you need to interact with the software, then you're going to be completely lost, and deciphering what that missing stair is really, really challenging."
In our casual conversations in OSS we found that getting lost is fairly common despite your knowledge of norms and processes and skills. OSS and by extension Science and Research OSS is most often built by a single person or small collection of people that cycle through intensity of activity on an OSS project. These are the people that are likely to have the most comprehensive knowledge of operating that OSS and even if they document incredibly well, there will be quirks in that OSS that others will not understand until they ask. This is where Usability can make a difference. In the period of time where the configurer of the OSS are making judgements and assumptions about general use using their own experiences as a benchmark which are often highly knowledgeable benchmarks.

Adjacent to these findings about how those that build and maintain scientific and research OSS describe an ideal user is a subtle, almost missed discovery about how usability is described by the scientists and researchers versus how it is described by those that work on usability regularly, such as designers.
Usability was often described as ‘how I want people to use the OSS tool’ or ‘how will we present our tool to users’ as well as questions such as ‘Can users see, understand and use our tool?’ and ‘How the user is viewing the system’. These phrases are subtle but suggest that the tool builders are delivering usable software to the users in the ways in which they believe are the most appropriate ways to use a tool. Put another way, there are ‘most correct ways’ of using the OSS tool and improving the tool’s usability means getting as many users as possible to use the tool in that most correct way.
A designer working out of a university based lab across multiple OSS projects describes the sometimes rigid and protective way that OSS projects can suggest users use a tool.
“An example is when they have this very specific queries as a way of querying the data. And either you don't, you know, or you don't. And it's, it can be simplified quite easily from a design point of view by using filters that are self explanatory, for instance, but they it's just, I don't want people that don't know about the subject to search the way to find out the way it's, yeah, as you say, it's way protective.”

Designers however, describe Usability as a process of working together with multiple people involved in the building of the OSS software to best understand the ways in which users want to and will use the tool and ensuring that these workflows and multiple ways of a user achieving a purpose with a tool (in SROSS’s case, solving a scientific or research problem where the OSS tool is part of that solution).
A designer who worked on integrations for a data cleaning and transformation OSS project describes briefly what usability means within SROSS projects which includes not being beholden to commercial and industry ways of performing usability improvements.
“Usability, I would say, is the job of designers working together with the tech team, so developers, product managers make that as easy as possible for users within the bounds of the project. So I'm also the kind of person who I would never advocate for this kind of old school, Silicon Valley thing like, don't make me think. I don't believe in that.”
Like similar conversations, these multiple user pathways tend to make SROSS projects nervous about resources, prioritization and scope of usability improvements.

Usability is mostly understood by Science and Research OSS contributors and builders only when the OSS is actively being used. Very few SROSS contributors and builders think of the usability beyond the act of using the OSS.

This finding was more of a gap in the conversation than an explicit conversation. Raised specifically by one project when discussing what usability is they began to explore strategic decisions made by the members of the project and the difficulty they’ve been having as a project in making clear, informed strategic decisions that reach beyond the maintainer teams own perspectives and understanding.

The lead developer maintainer of a 10 years mature OSS project that focuses on data analysis of machine learning in Python states *“People understanding the limits of the scope of the project — and why. And people using libraries correctly (within scientific methodological limits). There is a risk of using the tool incorrectly and then pursuing a line of inquiry that leads them to an improper result.”*
This conversation quickly began to describe the parts of the SROSS tool that were ‘outside’ of the functional OSS tool. The decisions before using the tool that the user makes, the ways they prepare their data, the research questions they approach the use of the OSS tool with, the hope and expectations they have of how the OSS tool might process and present data as well as how that output from the OSS tool informs the next use of the tool is all part of the wider user experience of the OSS, the service design of the OSS tool.

As this conversation with the participant concludes, our researcher notes in the progressing conversation that *“[The OSS project] participant seems to be thinking about the user problems that occur 'before' the tool/software is used and is frustrated/concerned/confused on how to tackle those problems if to tackle those problems”.*
This is an understandable concern on the part of the maintainer, how to better understand and build for the parts of the experience that comes before and after the use of the OSS is a complex usability question for someone not familiar with usability approaches.

Science and Research OSS misses out on the ways that dedicated Usability efforts could improve their tools and services in ways that they themselves express, but don’t yet realize is connected to Usability.

Developers and maintainers of early stages, low user numbers Science and Research OSS often spoke of answering email requests for help from users as a joyful activity. They described the act of directly helping and addressing someone's problem as a positive in terms of solving a specific problem for a person as well as having some clarity on how to improve usability or documentation generally. Developers here are doing customer support informed usability improvements. However, there were projects with significantly more users or different kinds of science and research fields that spoke of the support requests in a way that was time consuming and not efficient uses of their time.

Here a participant that was a computational scientist that now maintains OSS for data science projects running multiple components in Python describes the volume of questions the project receives and the desire to make this easier for those people asking questions:

“Leads to lots of questions from users that if we, if we could figure out how to do it, then we wouldn't spend as much time kind of answering basic questions over and over again. Right, software would be easier to use for people and that would kind of free up more of our time to actually work on developing and further as opposed to helping users do things that in theory should be simple.”

They describe a reality where if the software made sense to users then that would free up time for other work and priorities for the OSS.

They go on to describe the complex nature of the OSS tool *“What we're doing is kind of complicated, in some ways. And we try to make it as simple as possible, and how we encapsulate or abstract the complexity in a way that people can understand.”* and how the efforts on the documentation have helped *“we have gotten a lot of people that have been very happy with our documentation. Which is kind of weird, because again, there are people that definitely have problems doing things that we think should be simple”.*

This individual is performing usability improvements in the way that makes sense to them that they are trained in, writing and updating documentation and responding to requests for help. The alternative method that is not served here is what they suggest in the first quote which laments around ‘if the software was easier’.

Conclusion

We can clearly see from our research that there are ways that Usability is being practiced in Science and Research OSS, yet those teams without access to usability and design resources and experts hit a wall at certain points in these projects that they struggle to self-serve and problem solve out of.
The appetite for better usability is best described by a community manager supporting multiple SROSS projects.
“What I learned from all of that, though, is that there is a huge desire for people to improve the usability of their software, and then acknowledge that there's not a lot of forethought into what the trajectory of the software project will mean in the context of user accessibility later on. Usually, the thought process is how can we incorporate this feature as quickly as possible. But that being said, people do want to make the user experience better. And there's a huge desire to do that. But figuring out how to make that happen is a much bigger and harder question.”

If we can better communicate the need for usability earlier in the lifecycle of Science and Research OSS and assess their future needs better, support with educational resources that are directly applicable to the kinds of Science and Research OSS that exists and provide that usability support at the moments when the SROSS are ready (and just becoming ready) for support then we can mitigate these points where SROSS projects get stuck and spend resources writing additional documentation and responding to support requests without incorporating those support request changes into the OSS tool.

Our recommendation/s:

- **Make general OSS culture, infrastructure training and knowledge available to designers:** Ways for this knowledge to be collaborative and practical in nature could go beyond existing ways of collecting documentation and information and expecting passive consumption.

- **Make design, usability, accessibility training and knowledge available to Scientists and Researchers building OSS:** Ensure this training/knowledge understands and speaks to the elements that cause anxiety for OSS projects, such as prioritization, relevancy, industry norms and how those can be guides instead of ridgid must haves.
- **Document, share and create more opportunities for practical application of design and usability in Science and Research OSS:** We often heard that these concepts don't embed and become useful if they remain abstract and without specific, clear application to the type of Science and Research OSS that a person is maintaining.
- **Developing a series of assessments for Science and Research OSS tools:** To help them understand (before they are built and coded) how likely they are to go beyond single digit user base. So that design and usability might be considered as early in the build and development process as feasible.
- **Communicate how improvements to Usability can reduce time on maintainers:** Maintainers and developers spend writing detailed, step by step documentation guides and answering 'customer support' like requests when users are stuck can be incorporated into Usability improvements so users are not 'fixing' errors/misunderstandings via docs.

Prioritization

Usability, a competing priority?

Introduction

Like any other software, prioritizing what to build and who to build it for is an important aspect of directing development and maintenance of open source science & research software. Project leaders and maintainers strongly consider users when it comes to prioritizing what to build.

Many factors influence what leaders and maintainers prioritize, such as:

- the way users engage with a project,
- project funding, such as grants,
- the tools used,
- and the contributors involved.

Fixing and Adding Features are Prioritized Over Usability

Usability or user experience almost never take precedence over fixing or adding features. Scientific and Research OSS projects prioritize bugs or issues that impede a user's ability to operate or use the platform, followed by features and functionality above anything else. We consistently heard across projects that design and usability are not needed as much as fixing the breaking changes. Participants said that usability is "important" but not a "priority". For example, one respondent who identifies as a designer said:

“UX/UI is not seen as integral to the process. I don’t think it is valued less but not important enough to plan around or prioritize.”

User experience and usability tend to be seen as less valuable and irrelevant for scientific work and thus are not prioritized

Some projects perceive design as “prettiness.” For them, design is not seen as furthering the mission of science and as such because of which its design takes a back seat. Alternatively, design is seen as optimization or beautification of a feature which the SROSS community feels is a secondary activity and can be done later. It is the norm for projects to skip past any design stage.

One of the surfacing reasons for UX/UI to take a back seat is to make the most of contributors’ limited time, which is not always in abundance be it salaried staff or volunteers. While this would usually call for an early design involvement, in order to ensure, with low resources that the SROSS is built user-centered from the start, that is not the common case. Typically in commercial/non-open source projects that are resource constrained, design is more likely to be included earlier in the product development stage as a robust and comprehensive design research process helps product teams know earlier which features of a tool to de-prioritise based on user needs. In open source, because design is perceived as either resource intense or the project lack the access to design knowledge or designers, it is skipped and can therefore lead to a longer and less user-informed product development process by way of perceiving design as superfluous to immediate needs.. Typically in commercial/non-open source projects that are resource constrained, design is more likely to be included earlier in the product development stage. A robust and comprehensive design research process helps product teams know earlier which features of a tool to de-prioritise based on user needs. In open source however, it is skipped because design is perceived as either resource intensive or the project lacks the access to design knowledge or designers, and can therefore lead to a longer and less user-informed product development process by way of perceiving design as superfluous to immediate needs.

Projects consisting of UX staff are able to weave a layer of usability into the fabric of the software. And yet, there are projects that are unable to prioritize usability despite having UX staff. Some see design as an ever evolving, recurring cycle which gets in the way of shipping a usable product to users. It is something that “would change later if designed now”. Could it be that as a software community, we haven’t arrived at a clear language that explains how prioritizing usability and design will have positive impacts to the core of scientific work.?

Prioritization techniques in Scientific OS Software

Scientific OS projects adopt various techniques to prioritize user feedback like polling, voting, or discussing the priority order directly with users or clients. This is facilitated by periodic community calls, as well as communication and tracking tools like Github andGitter. However, there's no standard processes, decision making framework for prioritization in scientific OS product development.

There are several factors that impact prioritization:

- Users influence priority by being "involved", "active", "loud" or even simply by the nature of their public reputation.
- Contributors also drive prioritization in their own way whether they fix a bug, add a feature, add documentation or examples. Since they volunteer their time, they prioritize based on their personal interest.
- Client and grant relationships affect how user feedback is prioritized and implemented. A grant may prioritize certain issues over others.
- 'Urgent' and 'easy to fix' tasks and issues are prioritized. We found in a few interviews that an email reporting an easy or quick 'bug' or a 'problem' is prioritized because it can be tackled soon, as opposed to assessing whether or not it's broadly applicable to users or the solution they have is the most human/user centered fix for the bug/issue.

Conclusion

There is an awareness of accessibility and usability in scientific OS projects. In general, contributors voice the intention to prioritize meaningful design of the software and demonstrate openness to include design into the process, but they lack clarity to do so. This shows that UX is not seen as bad for the project but also there's less action on it. Sometimes, it can even be a fair rationalization to not prioritize usability when the project funding and sustainability directly depends on realized features for the software and not really the effectiveness of its use. It may be a long journey to normalize usability as a part of the scientific OS development process and not a competing priority. Establishing the purpose and impact of user experience design will be a great start.

Our recommendation/s:

- **Prioritize design and usability of the software from the beginning:** By weaving UX/UI into the design and build of software, project leads and contributors will save time later when responding to adoption and usability issues.
- **Create a clear and shared vocabulary around design and usability in OSS:** Grounding the definitions as a critical part of the build process would help shift perceptions and help project leads and contributors prioritize UX/UI.
- **Conceptualize and design the feature exactly how it needs to be built:** Since developer time on OS is limited, it is more efficient to design and then write code for it.

Accessibility

Introduction; accessibility as understood by Science and Research open source software and the self identified designers of this study.

Accessibility, as we found out during the course of this research, has a great many interpretations that have evolved and grown over the terminology origins and purpose. Originally understood as ways to ensure people with physical impairments and disabilities could access the same services as able-bodied folks through legislation such as the Architectural barriers act of 1968 in the USA through to various global technology and digital accessibility acts into the 1980s and 1990's, accessibility is now commonly understood as the ways in which services (online/digital and otherwise) are made accessible to people across the spectrum of physical and cognitive impairments and disabilities. This study, however, uncovered the nuances that people contributing to and building Science and Research OSS have when it comes to how accessibility is described and meant. In this section we'll also cover the differences described by the designers in Science and Research OSS as well as offering our own perspectives.

Accessibility is understood as making the Science and Research OSS tool more readily and easily available and usable by scientists and researchers that cannot, or never will 'be coders' as well as those that have historically been locked out of OSS (marginalized identities).

This understanding of accessibility is commonly spoken about in reference to building interfaces and GUI so that the SROSS might be operated through interface clicks, button, typing and menus as opposed to much SROSS which is operated through the command line or the terminal through a series of functional code commands.
A core developer at a bioinformatic platform speaks to the kinds of users that they have that are not always coming into the project with a level of coding knowledge that could be advantageous for them.
"A lot of the folks who are using our software are not trained computer scientists, or even like trained data scientists, a lot of them are biologists, who have picked up some command line software skills or a little bit of Python skills. And so keeping records can be challenging."

This supports and connects to a value that we discovered throughout the conversations about Science and Research that upholds the process and practice of visibility as connected to replicability and reproducibility in order for the SROSS to participate in peer review-like processes. Without the ability for other researchers across the specific research field and beyond to access, use, replicate and potentially modify then the SROSS does not meet criteria for others to reproduce the same results or processes and therefore has not kept within the academic/scientific process of peer review.

A core developer at a bioinformatic platform reinforces the importance of scientists and researchers being able to trace another scientist or researcher's process back to replicate.
“Users should always be able to figure out exactly what they did, you know, or somebody else should be able to figure out what the user did.”

A data scientist working in a university lab working on tools to improve human health speaks to how they make sure that the code they write is accessible to as many people that could do good with the tools and code as possible.
“Because we have the entire data collection system, we are trying to make it available for other researchers and eventually to pediatricians…so all of the code that I write is released publicly along with as much data as we can release on the vulnerable subject as well. And within my field, there's a lot of people that would use our system that aren't computational scientists. So all the code that I write, I really try and make sure that it's as accessible as possible for someone who might not have coding backgrounds.”
This same participant goes on to speak about the connected subject below, that many of our participants spoke to around open science and research being ‘for the common good’ and if it is not as accessible as possible, it is not leveraging the opportunities to do good in the world.

Another way that Science and Research understands and emphasizes accessibility as important is that by making SROSS tools more readily available and accessible to more scientists, researchers and potentially other audiences, they uphold open science and knowledge as a ‘public good’ where anyone might have the opportunity to contribute to a critical scientific discovery.

A topic that recurred in our conversations with projects was the fundamental reason why Science and Research exists, in order to advance and understand the universe around us and come to better understandings about it and it’s problems. This, through the creation and maintenance of OSS, is, in theory and possibility, made open and collaborative. Here is where accessibility intersects with the purposes of Science and Research, where the more accessible the open science, research and tools are, the more possibility there is for crucial findings to be discovered and replicated.
A developer for a bioinformatics platform states that *“I think by making the methods open, we can just advance more quickly to address advanced science more quickly and address global problems more quickly.”*
Here we can see the hopes for open science and research laid clearly and it is also here that we might infer that the more accessible those open methods are, the better the advancement will be.
The is underscored by a comment from a developer of a data format unification project where they describe their past as a scientist, now a developer and how they focus on making sure that the scientists and researchers can access the tools in ways in which are accessible to those users, in order to solve their scientific and research problems.

“I'm no longer a scientist...But I understand how their time is valuable and how hard it is for them to focus on their science. So my goal, as a former scientist, who does code now is to take the code out of the way, like, you don't have to worry about this hurdle of learning, programming and learning how to solve your problem.”

Lastly, we cannot forget that science and research for the betterment of humankind and the environment does not exist without the financial and government structures that support and enable it. A designer who works across multiple astrophysics projects speaks to the factual reality that many of these science and research OSS project are publicly funded in some way, and have a responsibility to the public regardless of its immediate relevancy or clarity to the public
“We're all publicly funded. So all the data needs to be accessible by anyone, but primarily by research scientists.”
The recognition here that yes, the priority is to other scientists and researchers, yet ultimately if the science, research and OSS are not able to be accessed and used by them, then it is less likely to be able to contribute to the public good, the advancement of scientific knowledge and understanding.

Like some other aspects of OSS culture, Science & Research OSS tends to share a view that accessibility also includes ‘free’ as in OSS is freely available to access if you have the means to and freedom to configure and modify, again if you have the means to.

This is not an aspect of accessibility known by people that have not embedded in the OSS cultural movements deeply and therefore designers did not mention this aspect of accessibility in our study and generally in our experience, designers only ‘design’ for OSS with this in mind once they have been included in these cultural conversations in OSS.

This is a complicated topic across all OSS. With opinions differing on what is the responsibility of the individual that wants to be part of the OSS and what is the responsibility of those that build and maintain the OSS to ensure what constitutes free beyond the descriptions of commonly used licenses. Often these commonly used licenses haven’t considered the explicit inclusion of people that may not have a certain educational basis in computer science, coding, literacy, english or that they have reliable, safe and stable access to the internet and suitable hardware. Social constraints also apply in terms of time a person has to contribute if they have met the knowledge, infrastructure and tangible device requirements. OSS often values a certain level of proficiency and dedication to OSS.
An open source community manager supporting multiple open science projects speaks on the expectations that project may have when it comes to people that manage to meet the implicit barriers to contribution to an OSS project
“Accessibility is true if you're saying people have the opportunity to use it. But accessibility is not being achieved if the goal is to ultimately increase the number of people who can use it at a level of proficiency that sort of meets their a prior expectations”

Another developer and maintainer at a Python based project for sounds based data speaks about how they feel like certain labs are getting attention from paper publishers because of their implicit access to tools and infrastructure and that there are people in labs in other countries doing amazing work without tool and infrastructure access yet they are unable to achieve a level of recognition. They speak to how this has informed how they build their tool with this kind of accessibility in mind.
“OS and [their specific OSS tool] lowers the barrier of entry for labs that may not have access to infrastructure tools”
This way of building tools describes accessibility in terms of equitable access and begins to demonstrate the complexity that the term ‘accessibility’ (and later usability) has within the Science and Research OSS field. This complexity pushes at a tension within OSS at large about diversity, inclusion and ethics in open source which is largely uncovered by official open source licenses yet is a critical and polarizing topic in OSS. From a designers perspective, we could lean on some common industry belief that the more diverse and inclusive the users involved in the process of configuring and building tools, the more accessible they become to those marginalized populations. This might then facilitate new context to scientific and research discoveries.

Designers in Science and Research OSS understand and apply accessibility in a different way that could be described as aligned with industry understanding of accessibility as inclusion of people using assistive technology as well as peoples differing levels of understanding, knowledge or context upon encountering a tool.

A designer at a lab that works across multiple Python related projects speaks to their frustration that the tools are unusable by assistive technology users.
"So…accessibility makes me really mad that our software is so inaccessible because it's also used in so many fields. I've met too many people who have said, I just can't work in this field. Because you know, I use this assistive tech, it won't work with it, I can't do the job."
This person is advocating for the explicit inclusion of a user that could very well contribute critical findings or code to the SROSS or the next scientific or research breakthrough in their field yet they are fundamentally excluded from using the OSS and are required to find workarounds for their assistive technology or not use that tool.
Designers often speak of accessibility and user inclusion as designing simplicity and access for the most complex of user needs. You'll often make that tool or service accessible or easier for others. A common and popular architecture infrastructure example is that when pavement curbs were lowered for wheelchair users they were also then used by those with other wheeled devices such as pushchairs, bicycles and wheeled zimmer frames as well as those that cannot step up or down easily. Making the city accessible to more citizens and improving their quality or life and their participation in society.
Another key consideration for designers is what legislation or government guidelines need to be adhered to when designing and creating open source, digital public goods tools. These legislations, like many funders and institutions encouraging or stipulating that SROSS be open

source, are ensuring the accessibility of these tools beyond their immediate use, into the distant future where they may be accessed again after the creator and maintainers are no longer contactable.
A designer in a university based lab that works across multiple OSS projects details how this legislation ensured that accessibility was considered in the design phase.
“Legislation for accessibility in the public sector helped. So we had to talk about it to make more people aware. So it became an integral part of the design. Personally, I'm really glad it happened. And the more inclusive and the more accessible, the better.”

The same designer goes on to speak about a somewhat unique case for SROSS where an output of the research was to be presented to the public through a museum exhibition. Here the designer describes how the complicated and contextual information needed to be accessible to the public, who likely have varied levels of knowledge about that context yet still, many levels of interest, knowledge, age and understanding needed to be considered.
“[a project] was specific to linguists and historic historical analysis of that, but then it was presented in a museum so it needed access to the general public. So these are two almost opposite ways of presenting. Counting it because you have a completely different language. You have different conventions of recognising elements. So you have to cater for both.”

Conclusion

We can clearly see in the insights about accessibility that it is a diverse term that cannot be used to express a single meaning or application in Science and Research open source software. However, we can also clearly see that the ways in which Science and Research understand accessibility ultimately contributes improvements to the term in better expressing what it means to be accessible, inclusive in the science and research fields at large.

We see can see the spectrum of designers, developers and the roles and functions in between doing the difficult work in sharing their understanding of the term and accessibility in practice for the good of broad user types (from the designers) and for the betterment of science, research and open tooling (from the developers and other OSS roles)
Here we see evidence of a Designer working across many Python related OSS projects to be inclusive and collaborative in their approach to accessibility. Creating a space where non-designers are informed and work together to make accessibility improvements to an interface in real-time and show that it is not always a process that is laborious.
“[What] I have managed to do with the accessibility space is create contributing event workshops, where you come, I'm going to teach you an accessibility concept, you're going to practice it on our code base, and we're gonna clean it up and contribute it, as a group.“

There are many more positive stories about integrating accessibility into SROSS project we discovered through the research and these project set a great example for projects that may believe, as covered in chapters around the perceptions of design and prioritization, that making a SROSS tool accessible, means to ‘rehaul’ ‘refactor’ and undertake a huge time investment into development.

Here a lead developer on a collaborative ethnography platform speaks to how they've prioritized mobile-first ways of using the tool.
"One main priority = accessibility. Have a friend working on accessibility for [their OSS project], who helped a lot in that area to bring ideas and things and they're now putting it into the code. Since the beginning, [we] designed in a mobile-first approach, which helped a bit with accessibility. But it's not something you'd be [typically] doing on your phone because it's too much text to read, pdfs to add, etc. There are usability issues on the list to improve on mobile especially. Then it's not only about rendering the content but about screen readers and how they'll read the content, and how people can see the colors, etc. Have a lot to improve re: accessibility"
This speaks to an honest assessment of accessibility where mobile might not be the common way these tools are used and most project may think in similar terms that there is too much information, data and processes for their OSS tools to be effective on mobiles, however, for a user that only has access to a mobile, or limited access to powerful lab computers or can only use mobile internet this option presents opportunities for their research that is worth making happen.

Our recommendation/s:

- **Making explicit and clear space for designers to communicate:** So they can share how improvements to accessibility (and usability) of SROSS is ethically impactful to the overall mission of science and research's aim to better humankind and the understanding of the universe.
- **Designers in Science and Research OSS must learn how Scientists and Researchers understand and value accessibility**: These understandings can be very different but are no less valid. This understanding can enable the two practices to better define when which part of accessibility is being focused on and why.
- **Trace and document issues/work back to a clear accessibility criteria/goal:** This helps clarity and openness across a SROSS tool. Accessibility can be part of both design and development acceptance criteria.
- **Look to the projects that have adopted accessibility best practices as case studies:** Have these projects act as models for how to include accessibility could show the way for the projects that believe accessibility is complicated.
- **Involve people with impairments in the building, maintaining, coding and design of SROSS:** Forming a working group or community for those who identify as such would enable a stronger platform for advocacy of needs.
- **Enable people from marginalized communities to have a greater voice in SROSS:** Similar to the above, ensure marginalized folks can voice what ways they need their SROSS tools to be accessible could support them.
- **Helping SROSS and Designers understand critical accessibility legislation:** Legislation changes and adapts and knowing what this means as it comes into effect across multiple digital public goods platforms will help accessibility in SROSS.

Part 3. Design

→ [Chapter: Perceptions of Design]
→ [Chapter: Design Practices and Workflows]
→ [Chapter: Design work in the academic context]
→ [Chapter: Designers' Experiences]
→ [Chapter: Challenges and Barriers]

Perceptions of design

Introduction

Perceptions of design in SROSS vary widely across people and projects, as per their lived experiences, governed by team dynamics, decision making processes, the role design plays in their work, among other factors. Design is subjective, hence, there is no right or wrong with how it is understood. When asked what design meant to them, our research participants came up with a broad set of responses ranging from design being an activity that is centered around users and community, enables collaboration, brings clarity, is visual, promotes thoughtfulness, yet is sometimes ignored, treated as optional and is aspirational.

Design is thinking about users and community

Design enables users to find their way to get started or 'onboarded' onto the software then nudges them towards the right workflows. Design focuses on human experience, makes the software inclusive for all users, reaching even those without technical proficiency, ones who might not want to dive into the code - since writing code is a prominent method of interacting with a science & research open source software. Design process involves the broad stakeholder community from start to finish and considers different perspectives.

Design is all about communication by challenging assumptions and bringing clarity

One of the participants who identified as a programme director thinks *"that's where design is super challenging to researchers, because they already start with a solution to the problem they want to solve. Stepping back to the basics of design, means they have to completely undo all their assumptions in their structures."*[2] Assumptions may not only be present in the way the software is structured but also the language and terminology, which users may not be aware of. Assumptions may lead to disparity between how the software is programmed and how the users actually use it, which then is the design's role to bridge by *"[understanding] the nature in while*

[2] This quote is also highlighted in design work in the academic context.

people interact with different features of the software" and then offering choices in a visible, actionable and clear manner. Design is thus all about communicating by unpacking assumptions, using a common language, representing the true state of the application and presenting information that's easily accessible.

The visual appeal, aesthetics, attractiveness, prettiness and branding of the software are all seen as design

Design is often seen in terms of 'look and feel' of the application, and common terms used in this context are 'fonts' and 'colors'. While there are other perceptions of designs mentioned above, those who see design in the box of visuals and aesthetics, often do not bring up the other values of design. One respondent expressed that *"design was almost put into the box of like, design is branding. Design is decorative, it has zero relevance to our, you know, hardcore dev community. So we'll just put this in the box and like, leave it out there."* On the contrary, another respondent observed that the visual aspect of design is rooted in psychology, and operates from the psychology of interfacing with the users which implies that it serves beyond the cosmetic value. A further participant that performs a design function cross multiple SROSS project adds to these thoughtful and exploratory perceptions of design when describing the changes that design has had by saying that the perception of design have changed and evolved as compared to the past into more than being *"just colors, fonts, what kind of images we have in the layout"*.

Design is also seen as the planning part of developing the software, a preparatory step in understanding the need and structure of the software

This take on design has a technical angle to it. One of the participants who identified as a researcher and maintainer explained their idea of design as *"picturing how to structure the configuration files for our software, and also the outputs and what plots people are going to want to see. Trying to think about how people will use my software, how will I and others build on my software in the future."* Another participant who is a master's student mentioned that design is *"understanding what you need, before even delving into building the software".* The structure, components and architecture of the software involve more technical and programming skills than interface conceptualisation skills. This perspective goes on to show the broad range of definitions that design takes.

Design helps empowers and helps fulfill the purpose of research

All functions of software development should help the outcomes and the goal for which it's built. The ability for design to empower research was heard from only a couple of participants. *"Enabling researchers to use technology to handle, discover, browse, and present their data to the general public through design"* as one UX/UI designer participant mentioned, helps fulfill the purpose of research. An element of thoughtfulness and design does not just go into the process of research but also the research output, so that questions such as *"how is research applied? How do people engage with it? How do you publish it? How do you make it open?"* can be answered.

It is hard to engage with design experts because of the prevailing structures in open sciences and lack of frameworks, even when there is interest.

There is a perception that designers and scientists live in different spheres and that their metrics of success in SROSS are extremely different. Seeing something as ‘other’ or ‘different’ resists collaboration. As mentioned earlier, this perception piggybacks on design’s ability to only deliver cosmetic value through animations, billboards, advertisements and the like. This wall between science and design is built on the foundations of grant styles which is palpable in the remark of one of the researcher participants who said that they had *“never seen an academic grant include design, in fact people will look at you funny if you try.”* In a similar vein another participant explained to us how *“academic grants typically give [projects] money to hire postdocs and PhD students. So you cannot easily embed someone that comes from a design perspective into your lab as there's no framework for that.”.* There have been innovative attempts to include design by pitching it as the *“co-creation of project with the community”* and there’s significant desire to improve user experience but *“how to make that happen is a bigger and harder question.”*

Conclusion

There’s truly a wide range of how design is perceived. It is often considered synonymous to accessibility. It is perceived as making users’ lives easier. It serves by optimizing and making software beautiful. Sometimes it can even be challenging to put specific words to explain design. Being aware of the different ways people understand design can help in building a more consistent and shared language among the community. Awareness can also promote a more calculated decision on making design a part of OS development.

Our recommendation/s:

- **Building a shared understanding of design:** How it adds value to the project and what is expected from a designer will support research and enable the fulfillment of research objectives.
- **Artifacts can change minds:** Stakeholders can be quickly convinced, if artifacts are produced that enable SROSS’s purpose and work and not distract them from it. For example, prioritizing the design of relevant evidence-led features and critical users usability as opposed to perpetuating an untrue stereotype that designers are most interested in ‘pixel perfection’ and ‘button radius’ can enrich the collaboration and help the mutual respect between SROSS and design.
- **Design in SROSS can be a lonely activity:** Supporting team members who look after the design aspects will eventually produce better value.

Design Practices and Workflows

How do SROSS without specific design capacity practice “design”?

Projects' openness to design depends upon how they think of the impact and usefulness of new features being implemented.

When projects discussed how they approach incorporating design work into the maintenance of their software, many, understandably, spoke to a principle interest in the relevance and usefulness of the improvements. Some projects are concerned with prioritizing design and new feature work that will be more welcoming to a wider range of users and contributors, while others are more interested in the efficiency and relevance of the proposed update to the existing user community.

This ranged from assessments that presented as a kind of cost/benefit analysis, to concern that an improvement would markedly improve a user or contributor’s experience with the tool. These considerations have significant overlap with the governance and prioritization models that projects employ, as deciding upon scope and relevancy in OS projects often falls back on how decisions are made in the project at large.

“There's a bunch of these kinds of things where it's like, trying to get a new API through means convincing people like there is a reason why we should add this new function and add this new test…And that means that we have to convince people this is actually going to be useful.”

Projects use various systems to track and complete design work; some integrate more “typical” design workflows, while others utilize tracking systems like Github that mirror how they handle technical issues.

Following the fact that very few projects have trained UX/UI designers on staff, few projects referred to use of more expected design processes, like sharing prototypes and feedback on prototypes, unless they had a designer on staff or in a consulting role. More often, projects referred to having text-based conversations about design issues in community channels like Discord or in Github. These projects said that they handle design issues as they do technical issues, using voting systems and replies to track relevance and status updates.

Design issues are often competing for developers’ time with perceived “critical” feature implementation or basic software functionality, often without agreed processes for handover collaboration processes between designers and developers. Few projects mentioned paying specific attention when prioritizing issues to design concerns. Some were intentional about making sure there is adequate time and thought allotted for critical design choices, and these projects were those who acknowledged that it is easy for design work to be overlooked or "missed."

Most commonly, maintainers in SROSS projects understand design as a feature of their codebase itself: how can they make the software as easily maintained/contributed to as possible? Critically, some such maintainers practice the same concept but don't understand it as "design work."

"Design means making sure you have what we call clean code, so making sure that the code is readable, the code conforms to best practices."

Many programmers who maintain these projects are understandably most familiar and concerned with the design of their codebase. Logically, to them, "doing design" in their project means *"structuring your software in a way that others and your future self can understand and build on and work with."* Not only is this good for the tool and those working on it, but doing a good job at code design has an added bonus of making your project more appealing to potential contributors. These efforts help make projects "more welcoming" when potential contributors can understand the code and find the information they need to make contribution decisions. So while we may call these efforts a kind of usability practice for code contributors, few if any developers used the word "usability" to describe their code work. Many did, however, call these practices their "design" work. Outside of the code itself, participants referenced documentation as a critical aspect of this pipeline.

While software design is a critical piece of design work undertaken by these projects, too much of a focus on the user-as-fellow-programmer can obscure the utility of UX/UI and broader usability work within SROSS projects.

"There are users and then there are developer-users that are then maintainer-users, and then contributor users. And I think a lot about the different overlapping user types of open source."

Participants were clear that the concepts of usability and design within these projects are slippery, especially when users can also be contributors, and some (but not all) also have similar technical proficiencies to the maintainers. While we heard a lot about projects needing to improve language barriers between disciplines, users and developers, and designers and maintainers, there is also an interesting phenomenon that occurs when many of the users, contributors, and maintainers *do* "speak the same language": how can SROSS projects develop an understanding of the nuances between the design of the code and how users interact with the software – *even when* there is overlap between those two categories?

One designer who works with these types of projects shared that they felt there is a "misconception that" what they do "is software design," when their practice doesn't involve the code design at all. On the other hand, some maintainers are actively trying to bridge the gap between code design and more typical UI design. One project maintainer shared that they're interested in building a GUI for their project to increase usability for people who don't know how to code. In the meantime, they are choosing to structure the code in a way that almost looks like a GUI. This person explained that they *"tried to picture what people will need as far as inputs and how to structure the configuration files for our software"* and chose to structure *"input files in a way that almost looks like a GUI, where it's like, here's an item, here's an item, here's an item."* In a way, this maintainer is trying to create a GUI-like experience for users of their command-line dependent project.

Our recommendation/s:

- **Designers valuing academic processes:** How can we view 'peer review' 'replicability' and community discussion as critical to design and user-centered design work?

Design work in the academic context

Many of the projects we spoke to either are housed within academic institutions, funded by academic institutions, or their maintainers were trained in academic institutions. Academic norms, pressures, challenges, and benefits are deeply embedded in SROSS projects. As discussed in the Open Source and Open Science chapter, academia as a container for science, knowledge production, and the scientific method can be supported and furthered by Open Source practices. It can also present challenges to openness. This chapter explores further how academia's structures and culture impacts maintenance and design work within the projects we studied.

Academic bureaucracy and norms can impede work, especially design work or projects that rely upon community or user-driven processes.

One project we interviewed defines itself as a citizen science project, has an ethos of collaboration with its patient-users, and practices user-led prioritization and design within its maintenance structure. They shared that working within academic institutions can be limiting because of the liability and legal requirements around doing research with human participants, like IRB processes. These hurdles introduce more complexity to their work, to the point that there's *"this discrepancy between what you're supposed to fill out in terms of paperwork to make sure people are treated properly and what you actually want to do"* to ensure the community is driving the project. This manifests as timeline interruptions on a significant scale: *"we make this prototype, and then you need to go back to the ethics review and say, 'Okay, now we want to modify our protocol, because the community gave us this feedback.' If you're lucky it takes six* *weeks to get a thumbs up, or to be met with more questions.*" For projects practicing human-centered design within academic contexts, these delays could become prohibitive.

Similar structural challenges make it difficult to bring design expertise into projects. Multiple participants shared that academic grants often do not provide a framework for bringing in design professionals. One shared that *"people will look at you funny"* if you try to contract a designer with your academic grant. One designer we spoke to shared that the concept is so foreign that they often must *"find the backdoor"* and be officially listed as a researcher to be hired onto projects that want their services.

While the academic bureaucracy makes it difficult for design to be officially recognized and contracted for, participants identified a real opportunity for designers to help academia do its work better by providing strategic design of research practices and programs.

Some participants encouraged that from helping researchers understand the "users" of their outputs to full service design of OSPOs, designers are needed within academic institutions. We were told that *"design is super challenging to researchers, because they get in their head a problem, and they often have the solution already in it. Stepping back and going back to the basics of design means they have to completely undo all their assumptions in their structures."*[3] In other words, this participant articulated that design is a way for researchers, who may be caught within the cultural and bureaucratic thorns of academia, to disentangle their work from structures that aren't serving them, their projects, or the science, and start to build programs and projects from a more strategic and integrated perspective.

One place that design does appear to be happening in academic contexts is in student work. There is a connection between student interest and university emphasis on industry-preparedness that opens the door for design with students. Participants active in Open Source Program Offices at universities provided a critical insight into how design and open practices are incentivized, adopted, and grown within these contexts.

Design is more of a norm and more an integral part of software builds outside of academia & OSS – in other words, in industry, design practices are standard. At the same time, universities are interested in their students succeeding after graduation, often in the form of industry-readiness post graduation. Incorporating design practice into student work builds professional competency. One participant noted that there's a tension between career

[3] This quote is previously highlighted and noted in the perceptions of design section.

researchers as professors and job-focused students: *“students’ main goal is to walk out with something they can get hired with. They're really focused on the applied aspects of that. And the researchers are not in that world. They're not up to date on the most common practices, because they don't need to be.”*

Two of our respondents shed light on how that tension is being addressed to the benefit of both students and universities. They were the founding team members of their University’s Open Source Software Center, which started less than a year ago, in July 2022. It is a club/organization where graduate students lead open source projects. While it started informally, with a professor leading it, it has already grown to have a VP, secretary, logo, and trademarks — with high demand. The idea came from a desire to give undergraduate computer science students a practical use for their capstone projects, while providing them with some more industry-like experience. That experience includes practicing more “typical” design work: usability tests, prototyping, etc. The students offer their services to science professors who need help building their research tools, and all of the projects are open source. Everyone wins: the scientists get help with their tools, the students get a practical and fulfilling use for their work with real users and “clients” to work with, and, as a participant in our research said, they are “giving back to the open source community.”

Designers’ experiences working with SROSS projects

Intro

Designers already work on open source research software, although this is not a common role. When talking to these designers, we learned about their methods to make design work in the structure of the projects, and how they manage their professional identities in a context in which design is not expected or seen as essential.

Designers in projects often need to justify design and educate others about the need for it to get buy-in

If there are designers in teams working on Open Source Research Software, they are often tasked not only with their design work, but also with educating the team on how design is done and why it's relevant. This is needed, since design is usually a late addition to projects, after other practices, like collaboration on code, have been established. Without educational work, design would remain an activity that is not seen to be relevant in practice.
For this, designers need to balance both making design an activity in its own right while not being perceived as threatening or opposing existing practices in the project. Designers employed different methods to do this:

- Presenting design as a “blank slate” without preconceived opinions
- Suggest design activities that do not need a lot of resources, like lightweight usability testing of existing ideas

- Demonstrating the usefulness of design in the context of an ongoing project
- Getting involved in existing discussions rather than opening up new ones

All these methods involve being not threatening and working in existing structures to build small wins that demonstrate that design can be a helpful contribution.

Designers also tried to manage their identity and distanced themselves from values and activities which are seen as “bad ‘in open source projects: “Polishing” (iterating in small steps with strong care for details), “superficiality” (strong care for immediate visual aesthetics), “maximizing clicks” (tracking user actions and compare them to goals) are such activities do distance oneself from.

Still, many struggled with achieving a mutual understanding. Designers wished for a “common language” which they tried to work towards by explaining terminology and educating team members. However, even shared terms did not necessarily translate into a lived design practice when working on the software.

Ideas about open source development and the design process can easily come into conflict. Design, with its abstractions and explicit negotiations is thought to happen before building software, as a participant said they *“…needed to educate about design as a step in the process before you start building things”*. This can be at odds with a code-focussed culture in open sources software: *“…it was also a learning process for me to understand that this open source community won't do what the designer tells them to do, they'll just do whatever they want to do.…they will try to build that feature, that feature will not look like the prototypes look, but they will be there.”*

Despite their strategy to be non threatening to existing conventions, designers also tried to advocate to allocate resources to design work. Designers can’t cast their design into code themselves and thus need to secure support from developers. As one designer said: *“I have to, like, lobby if I want people to do design work that they're not already working on.*” However, with many other tasks that developers can do and find urgent, it is often difficult to get the design-related work done and convince others to prioritize it: *“…I don't think that they think it's valueless, but it's definitely not important enough to, like, plan around or to like, take action on.*” The reason for not focussing on design was usually framed as either a constraint of “time” or “resources”.

Late introduction of design often needs risky and resource-intensive changes

Improvements to design need code changes, and changing or adding code can lead to ripple effects that influence other functions which then also need code changes.

Two people directly framed larger design changes as leading to *“rebuilding the entire project”* or the project *“[needing] a complete overhaul”*, for which *“the potential cost / risk is humungous”*. Even if the needed changes would not be as absolute as stated here, the trade-off still applies:

“It still works” and thus there is the question if it is beneficial to change and worth the change of established structures for an uncertain future. The prospect of finding out that such a large change is needed might be unsettling for a project. As a participant suggested: *“They probably want feedback on how to make their product better,”* but *“[might be a] little concerned to find out that there are major usability issues associated with it that will take a huge amount of unwinding and refactoring to resolve.”* and thus don’t engage with questions of design.

When projects were built around initial concerns of developers or scientists, their code is geared towards the changes that designers suggest. Thus, the designer’s suggestion might mandate larger changes to the code. Such deep changes to code of projects need a lot of effort and thus are seen as “too big/too late to fix” in many cases.

Such big changes are particularly difficult in an open-source mode of working: They need overarching changes which are, by definition, hard to do in a modular way. Yet working on isolated parts of the code makes it possible for single developers to collaborate efficiently.

An in-depth code change might also affect the existing community of users who are largely fine with the state of the software. The existing community of users gathered around how the software currently works. Doing substantial changes might lead to complaints or even anger of existing community members. As described in the section on “knowing users”, projects know the users who engage a lot with the project and write on mailing lists and issue trackers. These users already can use the software and would be affected by deeper changes that do not benefit them in the short term but actually might upset their workflows.

Conclusion

Designers in open source research software projects struggle to get buy-in for their suggestions– and with that also have troubles to legitimize themselves and their role. Often, a designer role seems to have been a concern introduced late in ongoing projects—if at all. This means technological and cultural structures that have grown established over years do not cater to designerly concerns. A designer can easily be seen as causing problems as their concerns might disrupt processes that have been working well for years in the past. Thus, when entering the existing structures derived from cultures of research and open source software, their main approach is to be non-threatening to existing concerns and practices, to demonstrate the value of design and to work towards a “common language” for mutual understanding. Despite these efforts, for many it remains difficult to reconcile design work with their project's practices.

Designerly concerns might also be hard to integrate into long-grown code structures that cater to needs of deeply engaged experts or coders. Code changes to cater towards the concerns of designers can be large and thus might be a risky investment of time or break compatibility with previous versions of the software. In addition, the existing community which is best known by the project’s creators does often not benefit a lot as they gather around the software as-it-is and thus can cope with the state of things, but might have workflows that would be upset by changes.

Our recommendation/s:

- **Expect challenges**: Introducing a design role in an established open source research software project is not trivial for the project in general and for the designer in particular.
- **Collaborate closely and clarify expectations**: As mutual understanding and a common language is a strong concern, projects who introduce a design role should plan for close collaborative work between roles, including time and activities to clarify mutual expectations and needs.
- **Plan for code changes**: Improvements in design usually need changes in code. This means that a designer can't be just added to an ongoing project in the assumption that they will provide a layer of design on top of whatever is currently done.

Design Challenges and Barriers

Challenges and Barriers for Science and Research OSS

When we asked people who work on Science and Research OSS tools to detail the challenges and barriers they have both generally and specifically toward understanding users, usability and design. Many of the general challenges and barriers have an effect on design and usability in the OSS and how users can be better understood. These challenges could be read as requiring a level of proficiency in how complex tech tools are built, how OSS is built and maintained as well as knowledge of the academic systems and science and research practice in these academic systems. This understanding, as well as time and the value of certain types of work (code as more valuable than any other contributions to OSS) can help us understand these challenges.

A widely accepted, inaccurate understanding of design is that it takes a long time, design and usability improvements will change too many things at once, or the OSS will have to refactor

One of the most worrying challenges raised by participants are challenges about the 'scale' of design and that design raises 'false hopes' for the users of the tool. Presumably, this means the tool will get directly better for specific users immediately. This is less of an innate challenge of design for complex OSS tools and more of a harmful public stereotype of design and designers. It also perpetuates a narrative around poor communication and expectation management on the part of both designers and OSS tool teams on what function design actually performs, and to what scale.
A developer at a data cleaning and transformation OSS tool speaks to these concerns about expectations and scale of design.
"[Our OSS tool] looks like it's 10-12 years old. Intense technical debt in the UI. So we've been very shy, engaging in the larger UX work because that's going to create false expectations in the

communities that we'll never be able to implement. Because there is so much we can't do. And if we want to do more, it's not a small step. But it's, it's really you take the front end, and you restart from scratch, and totally refactor, right"

In this next comment from a developer at an OSS microbiome bioinformatics platform, we see OSS projects notice when there are gaps in the expected user experience. The section that references 'if it's one user' or 'a couple of users' paired with the assumption that they likely 'misunderstood' a part of the tool and that the best course of action is updating or creating new documentation isn't inherently a bad course of action. Designers would argue that these couple of users are the users that decided to speak up on a usability issue and that there's likely more users that either have the same problem that don't voice it and move on with a different tool or solved their problem after battling it for a while. Regardless, the act of creating new documentation and improving the design and usability of these sections will not harm the other action, yet design and usability improvements are seen as complicated and less ideal than documentation that explains around potentially poor design or usability.
"When we run into a situation where users are not interacting with something like as we expected them to, we're typically you know, if it's, if it's one user, you know, maybe it's just something that was misunderstood if it's a couple of users, you know, maybe it's a gap in our documentation. And, you know, maybe it's kind of clear if you've got some context. But you know, there's some missing contexts that we're just that we didn't notice."

A common usability challenge is that the language or workflows used in the OSS tool might not be the most accessible to the various intended users. Designers may not be aware of specific terminology without the close collaboration with science and research experts.

One of the clearest examples of where designer and scientist/research collaboration is essential to successful usability improvements is found in examples that were repeated by multiple participants across job functions and OSS tool types. Examples of the highly specific nature of science and research OSS projects can be found in their use of terminology and language. A word or term might mean something different to two scientific or research disciplines and might convey a completely different message to people without that specific context knowledge (e.g. when science and research becomes of interest to the general public). We found this is also the same with icons or symbols used in interfaces. Designers and usability experts are able to suggest commonly understood words and symbols for GUI and other interfaces (e.g. the terminal commands and responses back to users) but there is a clear need for a process that involves scientifically trained users to test these suggestions and collaborate with the designers in order to achieve a word, phrase or symbol that best conveys a meaning across various levels of scientific knowledge and expertise. Remember, that many of these OSS tools want to encourage users from other scientific disciplines and simply using the most 'accurate' scientific terminology could render the experience unusable for scientists and researchers not yet familiar with that term.

A developer for a Python component composer program summarizes the difficulties in simplifying the complex nature of their tools: *“What we're doing is kind of complicated, in some ways. And we try to make it as simple as possible, and how we encapsulate or abstract the complexity in a way that people can understand.”*

A developer working across astronomy OSS tools as well as maintaining their own research OSS into financial data explains the dangers of making assumptions about users knowledge and usage: *“Ensuring that we're not making assumptions in the language that we use, when explaining to, to scientists about about how certain open software works, because, again, because we're programmers, it can sometimes be a problem that we might make assumptions, like maybe there might be certain terms that they might actually not be aware about.”*

A designer working in an university based lab across many Science and Research OSS on fixed term project times explains below the risks involved in communicating niche topics across many different levels of understanding: *“Icons and going back to conventions and very niche topics. Some symbols may mean something different to the general public from the very experienced user.”*

Designers who do work on Science and Research OSS are largely working solo or not in a space where they get much in the way of design peer support or design-related resources

In the commercial space, employed developers tend to outnumber employed designers in tool teams. In house designers are still growing alongside design agencies that are engaged on a case by case basis for design projects The outnumbering of developers to designers is also largely true of general OSS where there may be between 1-3 designers across well-supported tool with a stable and large user base. In less well supported OSS tools you’ll begin to see less designers present, unless they are themselves users of a tool or have a special interest in the purpose of the OSS tool. We spoke only to designers in the Science and Research space that were employed or paid in some capacity to work on the tools they designed for. The design volunteer contributor space is a complicated one that is under-researched and still growing. Popular design contribution spaces tend to be those that have some kind of design leadership intentionally growing the design contribution effort.

It is therefore not surprising that designers find it difficult to find other designers to work with and gain support from when working on Science and Research OSS tools. Some designers have other role functions that are design supporters, collaborators and advocators but not all designers have access to these allies.
Below a designer working across the Python OSS ecosystem for Science and Research describes that pushing for user needs to be implemented is more manageable when you have another advocate or designer to pair with on design work and to talk through ideas. Finishing with a comment where encouraging the tool teams to ‘stick with’ usability and user centered design is harder solo.

“I know one thing like when I do get to work on projects, with even just one more person in some kind of design, specialty, whatever that may be, like, honestly, even having one other person to bounce things off of has been really helpful. I've had others come in for projects, people have done all kinds of creative things to get some more designers in, and I super respect that. But in terms of sticking power, we have not.”

The benefits of having a multi-functional team inclusive of a designer are detailed by a designer working in a university lab across many different OSS tools on a project basis. They described positively their experiences in working collaboratively with their team and the support they received to advocate for users as well as develop their own academic and pedagogical growth.
“We build collaboratively, there is not a single project that is done by one or two people only. It's always a team effort from different remedies. So it's not just technological partners. This was also pedagogical and there were academic partners.”

The designer in the university lab was able to push against what the community manager working on a Python based data visualization tool describes below. They begin to describe the danger of homogeneous viewpoints being the dominant ones in tools that are meant to be accessible for many different kinds of users.
“So yeah, I mean, I think, like one thing we don't talk about but part of this is also like our steering council, it still has a very technical focus. And that helps speak to a farmer, geologist, a physicist …and also all white men in their 40s and 50s. Right. Trying to explain at some point, like why that homogeneity is not good.”
Science and Research OSS tools will always need the views of those described above, the consistently well represented people. But the challenge of widening access to more users and de-prioritising usability design will remain as long as those that remain focussed on the code-based technical work as the most important work.

Embracing change and communication is difficult when your structure of power within the OSS tool is historically those not familiar with design and have rarely seen benefits of design and usability improvements.

We’ve seen evidence from previous sections that the Science and Research OSS tool space struggles to broaden access to roles and functions that are not historically represented or responsible for founding the OSS tools, like design and usability practitioners. An effect mentioned in our research is that when similar kinds of people consistently are responsible for steering the direction of an OSS tool, it means that hesitancy to deviate from processes that have worked previously or feel comfortable in maintaining the status quo are not risked. Design and usability are only as risky when poorly planned. Change appears to be difficult to enact, a designer working across astronomy base OSS tools details a 25 year long process that, while rare, can be a source of friction and resistance to user centered design processes which often seek to challenge established norms and whether they remain relevant in the present.
“Old people who have been doing something a very specific way for 25 years, and they don't want to change. I haven't run into too much of that. But it's an attitude that I definitely see occasionally.”

Designers go to great lengths to establish foundations of communication. Without understanding the perspectives and knowledge that our collaborators have about what design is working on, designers would struggle to effectively communicate specialist and unique terms. Design can be viewed as a facilitative function, balancing the needs of users alongside the configurers of a tool.
“It's all about communication, finding a common language, because that's another barrier that I found. And it's, for me as well, understanding their topic and not, you know, I need to learn every time what they're talking about.” A designer, who works across multiple OSS tools from a university based lab, speaks to the understanding and shared respect designer and OSS tool builder need to work well together, ensuring that they understand enough from the scientific and research expertise to accurately balance the users needs with the purpose of the tool.

Conclusion

Helping Science and Research projects understand that design and usability work can be small, iterative and doesn’t require a ‘big rehaul/refactor’ process to be undertaken is the biggest impactful recommendation after understanding the challenges and barriers for projects. The key takeaway here is that even if users are having immense difficulty using a tool and are seen to be talking about that difficulty with the tool builders, unless the improvement to usability is seen as 'non trivial' which it often is not when it comes to design (this is likely because properly scoping design and usability work is difficult without a designer on staff or available) then it will not be done. It will likely be argued as a 'you need to read the docs more’ or ‘we need to provide training’ to help these users as opposed to making the OSS tool more usable. Many of these challenges come down to communication around time, expectations and scale of a design or usability intervention along with appropriate preparation and rollout time.

Below a designer working on a specific integration to a data cleaning and transformation tool, explains how they learned to adapt their design expectations (high fidelity prototype implementation) to a developer lead implementation that followed user research outcomes.
“High fidelity prototypes were never implemented, as they were designed. So they will, it was also a learning process for me to understand that this open source community will just not just, they just won't do what the designer tells them to do, they'll just do it, whatever they want to do. And they may or may not choose to listen to you, they will roughly stick to what outcomes came from, like the research. So users wanted x feature, they will try to build that feature, that feature will not look like the prototypes look, but they will be there.”
For the foreseeable future, designers will need to adjust their expectations of what design and usability looks like in Science and Research OSS. These OSS tools are not yet ready to match the commercial technology world in how design and usability is valued and trusted. Understanding what can be achieved within these complicated OSS systems is a good first step in making sure users are centered in these tools. If a level of design integration is able to be maintained, effort can then be invested in design becoming normalized and having better established positions for advocacy within these tool teams along with higher numbers of designers practicing in the Science and Research OSS tool space.

Our recommendation/s:

- **Support more pathways for designers to enter into Science and Research OSS work**: Designers and the peers that support them are few and far between. Those that are practicing design and usability in the Science and Research OSS tool space are unable to do so with the resources and support they need which is inclusive of simply, more designers doing the work and/or contributions.
- **Investing in design and usability resources, training or people:** These people can help Science & Research tools better scope design and usability work would help them to better understand what is doable within a given complex dependency infrastructure.
- **Invest time and efforts into design and usability examples:** Examples that communicate that it can be small, iterative and doesn't need to affect all or multiple parts of the Science and Research OSS tool. Understanding that design and usability improvements can be compartmentalized so that changes are not as 'scary' to users or as big a dev effort.
- **Build pathways for designers to sit at tables of 'power':** When it comes to making decisions on an OSS tool (or many Science and Research OSS tools) futures in accordance with users. Designers are 'lower' on the power scales than more valued functions, often unintentionally. Ensuring there are places where strong design voices can be heard and contribute to Science and Research OSS will build a user centered future for these tools.
- **Understanding and documenting:** Documenting the Science and Research OSS tool's dependencies and complexities both within and outside of the tool will help any design efforts to know the knock-on impact of design and usability changes. Ensuring that designers (and developers) are onboarded onto the effect that introducing knock-on features or changes has on potential future work.
- **For early stages of development:** Introducing a design system for any tools that have or will have GUI and ensuring that stays well documented. If a tool uses an existing UI kit or design system, understanding how and when that is extended to ensure that the tool doesn't get to the level of complexity and interwoven components that changing GUI's design, usability or UX negatively impacts on the system.

Glossary - words we use

Descriptive or theoretical terms

Terms on Research methods and research process

Codebook

Coding

Participants

Glossary - words from the field/research

COMMENT: Lines can be ordered e.g. in notepad++ via Edit/LineOperations/Sort by…

- **Acceptance criteria:** Typically added to a Github issue or a distinct piece of work as a set of rules to measure how successfully the issue solution/work meets certain criteria.
- **Citation:** Usually the reference to the source of an assertion. Being cited by other researchers is the major source of merit for a researcher. There are people arguing that software used in research projects should also be cited.
- **Churn:** Churn or “turnover” means that contributors leave the project. This poses a problem, since people need to learn about the project and its code before contributing work: They can’t be replaced quickly.
- **Community:** The people who are seen as part of an →Open Source →Project. Depending on its context of use it can include more or less people: In the most restricted use, it only refers to regular →contributors and long-time users who →contributors know personally; in wider uses it can also include users who are active on various project focussed communication channels or even users who never participate in this communication.
- **Contributor:** A person who contributes to an →Open Source → Project, usually by writing code. In recent years, there were initiatives to also apply the term to other activities that help projects including design, issue reporting, documentation writing or community organizing.
- **Design:** Planning or the result thereof. For software, there are at least two meanings of design: 1) The design of the user interface, so it is useful and usable which is usually done by UI/UX designers 2) The design of code, so it is easy to understand and to change, which is usually done by developers in general or system architects in particular.
- **Feature:** A functionality of a software that is useful for a user.
- **Git:** A software to manage and combine contributions to code; initially developed for work on the Linux kernel’s code. It frequently is used together with online platforms for code-focussed collaboration, most notably Github and Gitlab
- **Github:** An online platform for code-focussed collaboration of developers. Despite it being owned by a commercial company which is not sharing the platform’s source code, many open source projects use it to collaborate on code.

- **GUI: G**raphical **U**ser **I**nterface, today a common way to interact with computers via buttons, scrolling, and menus. Often used in contrast to a command line interface, which has been common up until the mid-90s for PCs, but remains an often preferred interface for software developers.
- **HTML & CSS:** Standards for writing code that defines content and look of websites.
- **Information architecture:** Purposefully creating a way to organize information on websites or other large information repositories. A bad information architecture (or a total lack thereof) leads to people not finding the information they need.
- **Interface:** Has two meanings 1) As graphical user interface or text user interface it refers to functionality of a computer program that allows end users to interact with it. 2) The way programmers can interact with larger parts of code written by other people.
- **Maintainer:** A person who is a contributor who stays with the project for a longer time and thus keeps it running, i.e. maintains it.
- **Open Data:** Sharing data created by research projects to allow others to replicate a research or to use for other research questions.
- **Open Source Program Offices or OSPOs:** Typically run by a company or an educational institution where there are staff dedicated to understanding how the program offering around open source in that particular institution or company operates and how it interacts with open source as a community externally and potentially how it contributes.
- **OS:** Shortened form of Open Source. Used as a descriptive adverb or a noun.
- **Product:** In this context, a software. The term is more common for commercial software that sells or rents the software.
- **Project:** Here used as (Open Source) Project, which refers to both the software created *and* its community: people are “part of” a project as active participants in its creation. Interestingly, these “projects” have no defined end, as it would be with “projects” in many other contexts.
- **Reproducibility:** A research result that can be “reproduced”, that is, replicated or repeated by another team of researchers. Within our research, this usually means that code and data are shared by the original researchers so that others can see if they come to the same conclusions. Reproducibility is a core principle in Karl Popper’s philosophy of science (Critical Rationalism).
- **Resources:** The distinct elements that are at the disposal or for the use of the project or team as a whole primarily, people’s applied skills, inclusive of their time to do those skills and money. The term is probably not used much by developers and more by (project) managers, product managers, co-ordinators, budget managers or community organizers.
- **Source code:** The text-based, human-readable representation of a computer program. From the source code, a running program can be created. →Open Source →Projects collaborate in writing the source code.
- **SROSS** (Scientific Research Open Source Software)**/ OSS** (Open Source Software)**/ OS *(Open Source)* / S&R** (Science and Research)**:**
- **User:** “The user” is usually referring to an imagined archetype summarizing all relevant people using the software. Thus, different people imagine “the user” in different ways. Particularly in the context of open source and hacker culture it can also refer to a person considered unskilled and being “the other” to the hacker. The problem that the term can

be nondescript and too encompassing is sometimes dealt with designers by using several more specific archetypes ("personas" or "user profiles")

- **UX, UI, UX/UI: UX** is the "user experience" of the use of a product, taking primarily the perspective of the user (and not the programmer or manager) when thinking about the design. *UX Designers* claim to design that experience, employing ideas from product design, user interface design and applied research. **UI** refers to the user interface, that is, what the user interacts with. There are different types of interfaces, but implicitly, *UI* often means a *graphical interface* with buttons and menus. *UIs* are designed by *UI designers*, who focus less on research or general product ideas than *UX designers*. Since *UX* is broad, but vague and *UI* overly focussed, these terms sometimes get combined to **UX/UI**: Concern for the user experience that, in practice, is often expressed via the (graphical) user interface. *UX/UI* can also be used as a general encompassing term when the exact distinction is not seen as important (as in "I never worked with UX/UI designers")