Making Software Work Sustainable for the Academic Research Group: A Comparative Case Study

Will Sutherland
University of Washington
willtsk@uw.edu

Andrew Neang
University of Washington
neanga@uw.edu

Charlotte P. Lee
University of Washington
cplee@uw.edu

Abstract

Studies of research software development have focused on how to promote or encourage the adoption of software engineering practices, but we do not have a good empirical understanding of strategies that researchers have already begun to take in order to integrate those practices into research work in sustainable ways. We conduct a comparative case study of two research groups in different fields, and characterize two approaches that they have taken to get research software engineering work done: practice integration and differentiating expertise. From these findings we argue that examining outcomes of change in research software development practice is critical for understanding sustainability and the ramifications of such changes for scientific work.

Keywords: research software engineering, scientific software, adoption, sustainability

1. Introduction

The problem of research software development is often framed, implicitly or explicitly, as a problem of adoption: how to get researchers to take up software engineering practices or best practices for software development. This has included codifying software engineering practices for easier adoption by researchers (Wilson et al., 2006; Arvanitou et al., 2021), assessing the kinds of work involved in such practices (Trainer et al., 2015), better understanding incentives and recognition structures for software work in the sciences (Howison and Herbsleb, 2011; Katz et al., 2020), identifying cultural barriers to interactions between researchers and software engineers (Segal, 2005), or characterizing researchers' development patterns independent of software engineering practices (Hannay et al., 2009). Given the recent growth of research software development roles in the sciences (Berente et al., 2019; Sims, 2022; Baxter et al., 2012), as well as greater uptake of software engineering practices by researchers, it is now possible for us to begin asking empirically how groups and collaborations are getting the "extra work" (Trainer et al., 2015) entailed in software engineering practices done. In other words, we can begin to move from asking how to promote these practices to examining how they are being taken up and used. Given the barriers, complex incentives, and alternative goals (e.g. Kelly, 2015) that exist in research labs and collaborations, how do these groups change their practice or reorganize in order to get research software engineering work done?

This question about changing software practices is straightforward, but requires departing from a model of adoption in a couple ways. Firstly, considering adoption focuses our attention on how engineering practices might be taken up into research work, but it does not further our understanding of how research work itself will change with the adoption of those practices. The implicit understanding is often that research work will be expedited: it will be qualitatively unchanged, but there will be faster or more correct results (Wilson et al., 2006). In this model software is a substrate, which supports scientific work, but does not touch the core of what constitutes scientific work. Some herald much deeper changes around software work and data-centric science (Djorgovsky, 2005; Brescia et al., 2017). More generally, prior work has characterized the tools and instruments of science as more deeply constitutive of the work of science itself than this substrate model might suggest (Hine, 2006; Paine and Lee, 2015). Looking at how research software engineering work does get done will require an examination of how the practice and organization of scientific work change around it.

Secondly, adoption can sometimes be taken as a generic process, in which different groups go through a similar process of taking up a codified set of practices. But different research groups and fields have different histories with software development work and are

likely to be going through different kinds of change around it. Efforts to understand the scientific context of research software development have focused on establishing essential or common characteristics across scientific work, such as that researchers often cannot express their requirements for software up front (Heaton and Carver, 2015), that they have a hard time establishing testing oracles (Kanewala and Bieman, 2014), or that research cultures favor research outputs over software contributions (Du et al., 2021). While there is some recognition that the sciences present a "varied software landscape" (Carver et al., 2022), codified sets of engineering practices are often understood as being for science writ large, and we still have a poor conceptual handle on the variety across fields, disciplines, or research areas in terms of how they approach software work.

In this paper we present a comparative case study of two research communities, one in the field of oceanography and the other in the field of radio cosmology, which have both recently undergone significant change in their work in order to improve the way they develop and use software. Through a qualitative analysis of interviews and observations we develop points of contrast between the two communities, and focus on change in practice and organization. Specifically we ask: *How do two different research groups change their work in order to get software engineering work done?* We characterize two processes, *differentiating expertise* and *practice integration*, which can help us understand the different approaches that these two communities took, and we connect these approaches with established practices and organizational environments in the two cases. Understanding the variety of changes the different research groups are undergoing around research software development work is useful in a practical way for planning and navigating these changes, but can also help us understand sustainability and long-term outcomes of these changes.

2. Background

Software has become increasingly essential to research in many fields (Carver et al., 2022), and is at the center of significant changes that are going on in scientific work, including larger reform movements around reproducibility and openness (Hong et al., 2022; Stodden et al., 2016). In recent years considerable effort has been dedicated to improving the reproducibility, sustainability, and correctness of software tools (Crouch et a., 2014; Mesh et al., 2014). These movements are usually approached as processes of *adoption*, where the goal is to encourage or facilitate the adoption of software engineering practices in the sciences, or better understand the barriers to this process. This has included codifying sets of best practices (Wilson et al., 2006; Arvanitou et al., 2021), examining cultural barriers to recognizing and rewarding software work in the sciences (Du et al., 2021), and characterizing cultural differences with software engineers (Segal, 2005).

Much of the research on adoption of best practices concerns science as a singular entity. This is in part because the research is motivated by an effort to understand what it is about scientific work that makes it difficult to adopt more structured or stringent software engineering practices (Heaton and Carver, 2015). In some cases, they point to characteristics of research work or process, such as the fact that researchers often cannot express their requirements up front, or that it is difficult to establish testing oracles (Kanewala and Bieman, 2014). In other cases studies point to aspects of scientific institutions or cultures, such as incentive structures which favor research outputs over software contributions (Du et al., 2021). These latter observations identify culture as a historically situated phenomenon, which can be intentionally changed (Katz et al., 2018), but it nevertheless looks for cultural or institutional attitudes which span all or most scientific fields. Both understandings of scientific process and scientific culture are looking for science as a monolithic phenomenon. While discipline might be a starting point for understanding variety in research software development, the situations of research software development will likely be shaped by a host of other factors. For instance, Howison and Herbsleb (2011) characterize the different relationships that researchers have towards software development work, including development as incidental to research work, for academic credit, or as a supporting service towards scientific work. Kelly et al., (2010) look at "technical" and "scientific" contexts as well as goals.

While some studies have looked at "successful" adoptions of software engineering practices in research groups (e.g. Trainer et al., 2015; Neang et al., 2023), the focus on adoption has thus far pointed our attention to places where software engineering practices have *not yet* been taken up in research work. Examining cases where such practices are being or have been taken up already can allow us to observe the outcomes or ramifications the adoption of these practices might have for scientific work.

Another recent stream of research on scientific software has looked at the emerging roles of research

software engineers. There are different characterizations of people working in this space. Baxter et al. (2012) describe a continuum from researcher-developers, who do significant software development in aid of their research goals, and research software engineers, who are knowledgeable about scientific problems but are focused on producing software. Berente et al. (2019) present a broader category of "cyberinfrastructure personnel", which captures a variety of roles involved in and complementary with scientific research work. Issues of hiring and retention of software-focused professionals in academia, as well as the professionalization and recognition of that work (Mundt et al., 2023), have been central topics in this discussion.

Studies of research software engineering and those on the adoption of software engineering practices in the sciences present somewhat different, although closely connected, visions for how software work can get done in research labs and communities. The problem of adoption focuses on a model where researchers themselves take on the extra work of more robust or engineering-like development practice, while studies of research software engineering focus on the possibility of coordinating software-focused work with research-focused work in one way or another. This difference is partially captured by Baxter et al.'s (2012) distinction between researcher-developers and research software engineers. These visions are of course not exclusive, and they are likely to be mutually interdependent. Nevertheless, it is valuable to examine how research labs in different contexts chart different courses through these different approaches.

3. Methods and Research Sites

Our study takes a comparative case-based approach, which is intended to provide an interpretive, naturalistic, and holistic account of software development work in two sites (Walsham, 1995). Our comparative approach is aimed at rendering clear, conceptual distinctions between historically situated entities, rather than identifying essences or commonalities across cases by accounting for variation (Ragin and Zaret, 1983). We develop the cases based on long term ethnographic engagement and analysis through qualitative coding of interview data (Table 1). Field notes were used to generate protocols, provide context for interview discussions, and to develop anecdotes for explication. Interviews were semi-structured and lasted between 45 minutes and 2 hours, and were recorded and transcribed using a transcription service. The interviews covered similar topics in both sites, focusing on our interlocutors' own experience with software development, how their practice had changed in recent years, the origin and development trajectory of particular software packages they were involved with, their opinions on software engineering practices, where they had learned those practices, and their interactions with other lab and community members around software artifacts. All names reported here are pseudonyms, including laboratory names.

Coding followed a theory-generative approach based on Charmaz's (2014) grounded theory. Specifically it involved an initial coding process, which was guided by our initial interest in comparing and contrasting engagement with software work between the two cases. This was based on the authors' prior observations and the recognition that the two groups differed in terms of the way that they accomplished software work, including the presence of software-engineering oriented members of the lab. However, the authors did not have a clear view of how to conceptualize this difference at the outset. This first coding process was therefore focused on difference in software practice across the two sites, but was also focused on producing codes liberally, without limiting codes to specific concepts or models. The authors also decided to code *across* the two sites, such that codes were developed with instances in both cases. The first two authors each coded documents from both cases, developing a single set of codes (using each other's codes where appropriate) using the qualitative coding software Atlas.ti. This resulted in 202 codes, which comprised both nascent analytical categories and in vivo categories. All three authors then consolidated this set through discussion, and by returning to instances of the codes. The authors considered various conceptualizations in this process, including comparing emphasis in different specific practices across the two sites (one focused more on unit testing or user feedback versus code review or code formatting, and so on), as well as comparing differences in the legacy codebases of the two groups. Overall the authors reframed the codes and the study in a couple ways: they 1) focused on codes describing *change* in practices or organizing, 2) focused the codes on processes or dynamics rather than on objects involved in those dynamics, and 3) focused on points of contrast between the two sites. The authors then performed focused coding around the new codes, which resulted in further refining the analysis to the two concepts we describe in the findings, differentiating expertise and practice integration.

3.1 Site Description

This study undertook ethnographic engagements with two field sites. The first of these is the Oceans Community, a biological oceanographic laboratory consisting of about 18 people, including Ph.D. students, postdocs, research scientists, a research professor, and the lab head. The group's work revolves around better understanding ocean microbe populations by collecting data at sea as well as growing cultures in a wet lab environment. Much of their software development work was done in Python or R, and involved building pipelines of sequential data analysis processes. The second author has been involved with the Oceans Group for 9 years, beginning in 2015. His involvement has included sitting in on lab meetings, multiple specific development meetings, interviews with lab members and collaborators.

The second field site, the Radio Group, works in a subfield of cosmology. Their work focuses on developing extensive signal and data processing systems to detect a signal emitted by neutral hydrogen during early periods in the development of the universe, which could tell us more about the universe's large-scale development. The lab consists of about 6 people, including Ph.D. students, one research scientist, and the lab head, a professor. Most of their software was written in Python, but they had at least one large pipeline that was written in IDL, which the group had used extensively in the past. The first author has been involved with the Radio Group for 5 years, beginning in 2019, and his engagement was similar in nature to the second authors' involving observations in lab meetings and development meetings, as well as interviews with lab members and collaborators.

Software work in both groups ranged from one-time and frequently reused scripts to larger open source projects, ranging from 25,000 – 120,000 lines in the Radio Group and 18,000 – 105,000 lines in the Oceans Group. Some of these projects were entirely developed within the lab while others were collaboratively developed with other labs. The most active projects had weekly contributions from 4-5 people, while some legacy projects were only updated every few months.

Both groups had, within the last 10 years, made significant efforts to improve the way they build and maintain software. Both had begun to implement practices that were new to them, including unit testing, continuous integration, and semantic versioning (among others) in the Radio Group and version control, documentation, refactoring, and requirements specification (among others) in the Oceans Group. Here we will describe these as "software engineering practices" for consistency with the literature (e.g. Heaton and Carver, 2015), although members of the two groups often simply referred to them specifically, or called them "best practices", "good practice", or "this good stuff" among other phrasings. While the goal of this study is not to quantify improvement that resulted from particular best practices, our argument is certainly based on the assumption that these changes in practice did make the groups' software work more sustainable. Our estimation of this was based on participant reports: interviewees explicitly stated that certain practices made the work more manageable or allowed them to continue maintaining the software, or they referred to earlier practices as regrettable or painfully difficult. It is also based on the demonstrable longevity of the software projects, a number of which have been actively developed for 8-10 years and have been used in producing numerous publications.

In both sites the "field", as constructed through the authors' ethnographic engagement (Karasti and Blomberg, 2018), was not limited to the singular lab, but also includes their larger multi-lab collaborations, who were often direct collaborators on their software projects and research work. We refer to the groups that are most central in our observations as the Oceans Group and Radio Group, while referring to the larger collaborations as the Oceans Community and the Radio Community. The last author negotiated entree for both field sites, has been instrumental in maintaining access, did some participant-observation in the Oceans Group, and contributed to analysis.

Category	No. Interviews	No. Interlocutors
Radio Group	15	13
Oceans Group	22	16
Undergraduate	1	1
Ph.D. Student	9	9
Research Consultant / Data manager	3	2
Research Scientist / Postdoc	12	9
Professor	9	7
Total	37	29

Table 1: Interviews by field site and position.

4. Findings

We describe two major dynamics, *differentiating expertise* and *practice integration,* by which the Radio Group and Oceans Group were able to implement software engineering practices in their software work. Differentiating expertise was a process of specialization, by which members of the lab began to work with people doing more software-oriented work, either in the lab or outside of it, in order to get pipelines or software packages built. Practice integration was a process by which researchers took up software engineering work on top of, or as part of, their day-to-day research work. These dynamics are distinct, but closely connected, and both were present to some degree in both sites. For instance, researchers in the Oceans Community were picking up programming and software engineering practices, but around more difficult development work they would get help from software-oriented members of the lab or collaboration. However, using these two concepts we can develop contrast between the two cases. The Radio Group engaged in practice integration, taking up software engineering practices into or on top of their research programming work, but did not engage in differentiating expertise to the degree that the Oceans Group did. Members of the Radio Group and the Radio Community were researchers who took up software engineering practices for certain development efforts and as overhead on their day-to-day research programming work. Members of the Oceans Group also engaged in practice integration, but they also dedicated significant effort to hiring or bringing on members of the lab who specialized in software development work. We describe each of these processes in turn, and use them to develop these two cases as distinct engagements with software engineering practice, with distinct outcomes and ramifications for research work in the two communities.

4.1. Practice Integration

Practice integration was a process by which researchers took up software development work and software engineering work on top of, or as part of, their day-to-day research tasks. Particularly in the Radio Group, this meant that researchers took on a considerable amount of work in software development-oriented meetings and performing tasks like writing unit tests, reviewing others' code, or writing tutorials for their software. However, taking on these practices collectively also created tensions around the disciplining, adherence to, and promotion of software engineering practices. Researchers in the Oceans Group also integrated software engineering practices into their work in this way, but their experience differed somewhat both because of differences in established practice and because a great deal of software work was taken on by others who had backgrounds in and were focused on, producing software, as described in the section on differentiating expertise. We will first discuss some of the collaborative tensions that arose for the Radio Group and then the differences visible in the Oceans Group's engagement.

Members of the Radio Group often discussed software engineering practices as a matter of discipline. People talked about trying to "do better" with specific practices, and while they often had a sense that they *should* be doing these things, they had to balance them against time or effort available. The Radio Community's research involved building shared analysis pipelines, which required contributions from a number of different labs and which would affect everyone's results. For this reason they had developed a strong sense of obligation to use certain development practices on certain, shared analysis software. But as Cliff, a professor in the Radio Community, describes, this was strategically done for particular software:

> *"So a lot of our other analysis codes are pretty heavily tested. And we hold ourselves to pretty high- even if they're not tested, we hold ourselves to a pretty high development standard, with the branch-pull-merge architecture and you know, that sort of software development protocol, but that's- that takes a lot of people in the loop, which is why we do it. But if you're just trying to get plots moving quickly… and we try to hold ourselves to that high of a standard for the first- especially for stuff running as part of the instrument on site. Either that or around the critical path to a paper"* (Cliff, Professor).

Contributions to software on the "critical path" would require researchers to write tests and tutorials and other such practices, but there were also exploratory kinds of work done in computational notebooks or scripts which were not held to the same "high development standard". Some software packages in the community had templates for contributions (pull requests) which had checkboxes where the contributor needed to indicate that they had written tests and tutorials for their code. This helped enforce practices that

individuals saw as important but laborious. In this sense members of the community understood software engineering practices as an onerous obligation, and they strategized or triaged places where they were needed across the community's different software tools in order to mitigate the burden.

Particularly because many of the Radio Community's software projects were collectively developed, encouraging or promoting software engineering practices was a collective effort, and could be a source of tension for that reason. Some members of the community were seen as proactive proponents of software engineering practices in the community, or were exemplars, who learned and implemented them in ways that others could copy. This was a kind of championing of software practice, which garnered respect from other members of the collaboration, but it could also become a point of collaborative or interpersonal tension where promoting best practices became a criticism of existing practice. Cliff also described the informal way that this criticism might happen:

> *"I mean, we sort of have like a kind of community marriage, you know. You take turns being the bad guy. Whoever, whoever happens to be most sort of, most recently burned by the particular problem pointed out so- so sometimes it's test coverage, sometimes it's tutorials, sometimes it's, you know, community stuff"* (Cliff, Professor)

Being the "bad guy" in Cliff's description captures how people could feel defensive against these kinds of criticisms of their work, but it also captures the fact that championing was not always a desirable activity, because sometimes it meant assuming the position of critic. Mila, a member of the Radio Group who was referenced by a number of people as a common source of best practice and development expertise, expressed some relief in an interview because she no longer was the only person taking on this role: *"I felt like I was sometimes the one fighting that fight. And now it's like, I don't even have to push that hard because other people jump in…"* (Mila, Research Scientist). The emergence of these roles and the collective navigation of the tensions that emerged around it was a novel organizational dynamic for the Radio Community, and one which they had to figure out in order to successfully take on the extra work of software engineering practices.

The two groups' processes of practice integration looked somewhat different because the two groups had different established practices around programming work in general. In the Radio Group, programming work had been a core part of day-to-day research work for a long time, such that older members of the group had done significant amounts of software development work when they were Ph.D. students or postdocs (although often without the more recently adopted engineering practices). Programming and pipeline development were expected and well understood kinds of work in the field, both in terms of planning student projects and timelines and in terms of mentoring and training. Researchers in the Oceans Group had a number of different technical skill sets, spanning wet lab work and the operation of instruments at sea, as well as data analysis, which, some years before our involvement with the site, had been done with tools like Matlab, Excel, or R. With the introduction of new kinds of instrumentation into the field (particularly metagenomic sequencing, but also instruments for measuring chemical composition and others) the scale and complexity of data required a large jump in the amount of software development work required, and particularly the development of extensive analysis pipelines. Kaci, a Postdoc in the Oceans Community, described how this was changing expectations for the field at large:

> *"But I would say like most of the postdocs in my field, don't have the coding skills that I do. And I think that sucks for them, because they're forced to learn it now, because data sets are just getting bigger and bigger, and you can't do them in Excel anymore, essentially. [...] But it's crazy how most PhD students these days know how to code. It's motivating for me to be like, wow, you've gotta keep up in some ways. [laughs] When I started, we didn't have these instruments that create this kind of data that we would need these kinds of skills for. So there's– when I first started there was no motivation for me to learn these skills"* (Kaci, Postdoc)

For students in the Oceans Group as well as in collaborating labs, R and Python had become increasingly popular (and expected) environments for doing data analysis amongst younger members of the community, and many described picking up programming either through tutorials online, through classes at the university, or from other lab members. This process was somewhat different than the adoption of software engineering practices, as it involved learning the materialities and the in-and-outs of bespoke software as a kind of tool; learning how different software packages could be put together and borrowed, how to assess the robustness of other software tools, learning how much time development

tasks would take and what the potential pitfalls were. Kaci describes how she and her advisor worked through this process:

> *"I think she thinks it's a little more plug-and-play than it is? And maybe for people who are more, like, experienced with putting together different pipelines, or different parts, different tools into a pipeline, it is more plug-and-play, but for me it's really not. It takes me a long time just grapple with what is it doing. [...] I mean, I think that, I don't know how [Advisor] feels, but I suspect that she has been surprised at how slowly that we've been able to get from an instrument to a story. I think she expected that to be faster?"* (Kaci, Postdoc)

Of the labs in the Oceans Community, the Oceans Group was particularly proactive about picking up software development as an analysis technique, as well as software engineering practices. However, there was significant variety in the Oceans Group in terms of familiarity with programming and software development work. Some Ph.D. students did quite a lot of it, while others' were focused on other skill sets and had to pick it up as a kind of overhead work.

4.2. Differentiating Expertise

Differentiating expertise was a process by which work within a research became more specialized, such that some members of the lab developed significant expertise in software-related work, including software engineering practices, while others had expertise primarily in particular research topics and other kinds of technical skill sets (operating instruments and wet lab work). Members of the lab were able to build analysis pipelines and accomplish data processing-intensive science by coordinating these different expertises as complementary contributions. The Radio Group had some degree of differentiating expertise in the processes of championing, described above, where some members became particular proponents of software engineering practice, but members all developed expertise in both software engineering practices as well as their research topics. The Oceans Group, however, was proactive and quite savvy about differentiating expertise, hiring or bringing on people (as students) who had significant experience with software development or whose primary object of work was building software, and making roles for them within the lab.

This differentiation within the Oceans Group was of course not discrete but took on some variety. Columbo, who entered the lab as a lab technician moved from wet lab work to doing "script jobs" and building shared infrastructure:

> *"So I started working in my sort of spare time in the lab on these little side projects where I'd write some Perl scripts in case it was something [Gina] wanted, and eventually that became more and more my job. [...] And I probably was a lab tech for two years? I think? I mean full lab tech, that's all I'm doing. Um, or maybe a year, and a half until I started to slowly, I didn't even realize what I was doing, but, you know, by taking on these little scripting jobs, and Gina kind of pushing me, to keep doing that because that's what I found interesting, and she had a need for it. Um, slowly, over time, I think, probably within three years, I kind of fully transitioned into really starting to do what I do now"* (Columbo, Research Consultant)

As has been observed in studies of research software engineering (Berente et al., 2019), there were not initially well-defined roles for software-oriented work, and people who did that work often had somewhat circuitous or serendipitous routes into their roles, like Columbo's. Valentino had a long career as a software developer before joining the lab as a Ph.D. student. Given his background he became a source of software engineering expertise in the lab, but his own work was at least partially oriented towards research outputs. Some younger Ph.D. students similarly became quite proficient in software development and engineering techniques and began to help other students with their software work. Another lab had a data manager role that also involved helping researchers with using Git and Github, and the Oceans Group also had briefer incubator-like interactions with computer scientists at their university. In this way the Oceans Group, as well as other labs in the Oceans Community, became a place of differentiated skill sets, and a place where people with different backgrounds entered and left to pursue quite different future careers. This dynamic was captured roughly by Valentino, who described himself as a "carpet dweller", because he spent most of his time in the carpeted offices of the lab rather than the wet lab, the "shiny side" of the lab. This two-sided distinction captures a difference in the day-to-day work of the lab, but it belies the complexity of career trajectories present there, with some people pursuing biology but picking up software development experience in aid of that goal, others who had some background in biology but who were primarily interested in software-oriented work, and those with experience in software development who were pursuing biological research.

Part of what was involved in this change for the Oceans Group was figuring out how to connect and coordinate the work of people with different expertises. Franz, a research professor in the Oceans Group who was developing a novel analysis pipeline for a particular instrument, attempted a series of arrangements with different people before finding a sustainable way to get more robust software work done. He initially worked with a group of computer scientists on campus, who designed a database system for him based on a consultation. Franz found the system to be overcomplicated and unusable, however, and so enrolled one of the students involved to help him rework the pipeline:

> *"So I work side by side with [Ph.D. student], which was great. Yeah we were sitting next to each other for a week, and I see– I learn so much, from some things to do, things not to do, so he helped me a lot, correct some habits that I had with, I'd tend to do too much in one function, and he told me no, you've got to cut different functions, instead one big function you do several, 2, 3 small functions."* (Franz, Research Assistant Professor)

For Franz this way of working was more effective in coordinating what he was trying to do in developing the pipeline with the expertise he needed to better design it. He contrasted this with his first interaction:

> *"So after that we decide to do a side-by-side, working really together instead of saying 'Hey, here you go, can you do this for me?' We are gonna do one meeting to explain what I want and 2 months later we're gonna see each other with the final product"* (Franz, Research Assistant Professor)

This "side-by-side" work helped by extending Franz's pipeline, but it also enabled him to learn and later maintain the code (a kind of practice integration). While Franz found this arrangement useful, the Ph.D. student was certainly not in a place to provide long-term support for the software. Ultimately, Franz was able to get a significant amount of Columbo's time dedicated to working on his instrument pipeline. As a permanent member of the lab, Columbo was in a position to provide both advice about development practice, but also to handle more difficult parts of the development (such as parallelizing certain processes) and to help implement software engineering practices, such as code refactoring, some tests, and better documentation. Figuring out how to make these kinds of collaborative efforts work, between people focused on software work and people focused on research work, was non-trivial and was a central part of making the differentiation of expertise in the lab work.

Differentiating expertise also complicated lab members' career paths. This was most apparent for those more focused on software work, who were occupying new roles and trying to pursue software-oriented career paths in academia, as described above. However, research-oriented members of the lab also found themselves needing to navigate career paths that were increasingly dependent on technical pipeline development. Bernice, a Ph.D. student, described how another Ph.D. student had become a "crutch" for getting her scripting or software work done, and that she felt the need to pick up the skills herself for the sake of her career:

> *"I know what the steps are but not in practice. So, it does… [Ryder] can be a little bit of a crutch for me. Because I can just be like '[Ryder] can you do this and run the code that you have already and then do it.' And he's nice enough that a lot of times he'll do it for me. But I'm really trying to be more aware of what he's actually doing and try to do some things on my own. But it would take me hours, days to reconstruct the things that [Ryder] can already do pretty quick. Yeah, I mean, the only thing that's scary is like [Gina] is always like, 'at some point you're gonna be alone. And you're not gonna have a [Ryder], so, you need to be self-sufficient.' So, she's like a big ... Totally fair and I get it. But, it's also hard 'cause there's always a deadline"* (Bernice, Ph.D. student).

For Bernice, there is a balancing act between relying on those with software-oriented expertise, and learning such development skills herself so that when she moves on to another position she will not be hobbled by the potential absence of such support. This was in addition to developing technical skill sets that members of the lab developed in doing wet lab work and operating instruments at sea. In this way the longer trajectory of researchers' career paths, the skills they would need to learn but also potentially the skillsets they would bring on when building their own labs in the future, were changing around the new complementarity of expertises.

5. Discussion and Conclusion

Practice integration and differentiating expertise represent two approaches to getting software work done in research groups and collaborations. These concepts can be useful as sensitizing concepts (Blumer, 1986) for understanding variety in the approaches that

research groups take to improving their software development practice. As noted above, the two processes were distinct but interrelated, and it is likely that a research group would leverage some of both in trying to improve the way they build software. They are also broad categories in themselves. For instance, differentiating expertise might involve bringing software developers into laboratories, working with external research software engineering groups, or collaborating with researchers in a software subfield (Howison and Herbsleb, 2011).

Our findings point to the importance of approaching the sciences as a plurality, where software work is situated in particular contexts with organizational and disciplinary dimensions (among many others). Studies of methodology tailoring (Campanelli and Parreiras, 2015), for instance, have attempted to account for aspects of context (such as culture, needs, and strategy), but more work is needed to account for the variety of arrangements for software work that can be accomplished in the sciences. Most practically, such research could inform heads of labs or software projects in making decisions about what arrangements (short term RSE engagements, longer term RSE staff, further carpentry-style training) to pursue given available funding, timelines, state of legacy software, and established practice. Such studies could also help research software engineering groups better understand the conditions for a successful engagement with different kinds of research groups.

Our findings also begin to characterize the different kinds of challenges that might come up for research groups in pursuing these different approaches to getting software work done. In the differentiation of expertise this includes the overhead of learning to manage, and promote the career of, non-research oriented software developers working in the lab, as well as streamlining processes for handing off or delegating work between research-oriented members of a group and more software-oriented members (or with members of a research software engineering group). In practice integration it included the interpersonal tensions that can arise in negotiating collective expectations about what software engineering practices are expected and when.

We have also attempted to highlight *outcomes* of the transitions that these two groups have undergone around software development work. For instance, the Oceans Group became a place with a greater internal heterogeneity of roles and professional trajectories, particularly in terms of members' different orientations towards producing scientific results or producing research software (Baxter et al., 2012). The Radio Group (and Community) developed a new sense of collective rigor around software tools, and established processes for when and where to expect or enforce new requirements on software work. These are important examples of how research groups *have made* research software development work sustainable. Moreover, these changes are not just the addition of a few practices to their daily work, but rather software is being enrolled in processes of reorganizing collaborations, change in understanding of vocation (Jackson and Barbrow, 2013), and the cultivation of an installed base (Hanseth, 2010) of instrumentation and computing resources. Given this broader entanglement, it would be valuable to approach change in research software development practice as a process of more holistic sociotechnical reconfiguration (Mazmanian et al., 2014), or otherwise a more constitutive change in scientific work and organizations.

Acknowledgements

This article is based on work supported by National Science Foundation grants (#1954620 and #1302272).

References

Arvanitou, E. M., Ampatzoglou, A., Chatzigeorgiou, A., & Carver, J. C. (2021). Software engineering practices for scientific software development: A systematic mapping study. *Journal of Systems and Software*, *172*, 110848.

Chue Hong, N. P., Katz, D. S., Barker, M., Lamprecht, A.-L., Martinez, … RDA FAIR4RS WG. (2022). FAIR Principles for Research Software (FAIR4RS Principles) (1.0). Zenodo. https://doi.org/10.15497/RDA00068

Barley, S. R., & Bechky, B. A. (1994). In the backrooms of science: The work of technicians in science labs. *Work and occupations*, *21*(1), 85-126.

Baxter, R., Hong, N. C., Gorissen, D., Hetherington, J., & Todorov, I. (2012, September). The research software engineer. In *Digital Research 2012*, Oxford, United Kingdom.

Berente, N., Ahalt, S., Bottum, J., Brunson, D., Cutcher-Gershenfeld, J., ... & Winter, S. (2019). The professionalization of cyberinfrastructure personnel? In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)* (pp. 1-6).

Blumer, H. (1986). What's wrong with social theory? In *Symbolic interactionism: Perspective and method*. Univ of California Press.

Brescia, M., Cavuoti, S., Amaro, V., Riccio, G., Angora, G., Vellucci, C., & Longo, G. (2017, October). Data Deluge in Astrophysics: Photometric Redshifts as a Template Use Case. In *International Conference on Data Analytics and Management in Data Intensive Domains* (pp. 61-72). Cham: Springer International Publishing.

Campanelli, A. S., & Parreiras, F. S. (2015). Agile methods tailoring–A systematic literature review. *Journal of Systems and Software*, *110*, 85-100.

Carver, J. C., Weber, N., Ram, K., Gesing, S., & Katz, D. S. (2022). A survey of the state of the practice for research software in the United States. *PeerJ Computer Science*, *8*, e963.

Charmaz, K. (2014). *Constructing grounded theory: A practical guide through qualitative analysis*. Sage.

Crouch, S., Hong, N. C., Hettrick, S., Jackson, M., Pawlik, A., Sufi, S., ... & Parsons, M. (2014). The Software Sustainability Institute: changing research software attitudes and practices. *Computing in Science & Engineering*, *15*(6), 74-80.

Djorgovski, S. G. (2005, July). Virtual astronomy, information technology, and the new scientific methodology. In *Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05)* (pp. 125-132). IEEE.

Du, C., Cohoon, J., Priem, J., Piwowar, H., Meyer, C., & Howison, J. (2021, October). CiteAs: better software through sociotechnical change for better software citation. In *Companion Publication of the 2021 Conference on Computer Supported Cooperative Work and Social Computing* (pp. 218-221).

Hannay, J. E., MacLeod, C., Singer, J., Langtangen, H. P., Pfahl, D., & Wilson, G. (2009, May). How do scientists develop and use scientific software? In *2009 ICSE workshop on software engineering for computational science and engineering*, Vancouver, BC, Canada. IEEE.

Hanseth, O. (2010). From systems and tools to networks and infrastructures-from design to cultivation: Towards a design theory of information infrastructures. In Holmström, J., Wiberg, M., and Lund, A (Eds.) *Industrial informatics design, use and innovation: Perspectives and services* (pp. 122-156). IGI Global.

Heaton, D., & Carver, J. C. (2015). Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology*, *67*, 207-219.

Hine, C. (2006). Databases as scientific instruments and their role in the ordering of scientific work. *Social Studies of Science*, *36* (2), 269-298.

Howison, J., & Herbsleb, J. D. (2011, March). Scientific software production: incentives and collaboration. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work* (pp. 513-522).

Jackson, S. J., & Barbrow, S. (2013, April). Infrastructure and vocation: field, calling and computation in ecology. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (pp. 2873-2882).

Kanewala, U., & Bieman, J. M. (2014). Testing scientific software: A systematic literature review. *Information and software technology*, *56*(10), 1219-1232.

Katz, D. S., McInnes, L. C., Bernholdt, D. E., Mayes, A. C., Hong, N. P. C., ... & Wilkins-Diehr, N. (2018). Community organizations: Changing the culture in which research software is developed and sustained. *Computing in Science & Engineering*, *21*(2), 8-24.

Katz, D. S., Hong, N. P. C., Clark, T., Muench, A., Stall, S., ... & Yeston, J. (2020). Recognizing the value of software: a software citation guide. *F1000Research*, *9*.

Karasti, H., & Blomberg, J. (2018). Studying infrastructuring ethnographically. *Computer Supported Cooperative Work (CSCW)*, *27*, 233-265.

Kelly, D., Thorsteinson, S., & Hook, D. (2010). Scientific software testing: Analysis with four dimensions. *IEEE software*, *28*(3), 84-90.

Kelly, D. (2015). Scientific software development viewed as knowledge acquisition: Towards understanding the development of risk-averse scientific software. *Journal of Systems and Software*, *109*, 50-61.

Mazmanian, M., Cohn, M., & Dourish, P. (2014). Dynamic reconfiguration in planetary exploration. *Mis Quarterly*, *38*(3), 831-848.

Mesh, E. S., Burns, G., & Hawker, J. S. (2014). Leveraging expertise to support scientific software process improvement decisions. *Computing in Science & Engineering*, *16*(3), 28-34.

Mundt, M. R., Beattie, K., Bisila, J., Ferenbaugh, C. R., Godoy, W. F., ... & Teves, J. B. (2023). For the public good: Connecting, retaining, and recognizing current and future RSEs at US national research laboratories and agencies. *Computing in Science & Engineering*, *24*(6), 6-13.

Neang, A. B., Sutherland, W., Ribes, D., & Lee, C. P. (2023). Organizing Oceanographic Infrastructure: The Work of Making a Software Pipeline Repurposable. *Proceedings of the ACM on Human-Computer Interaction*, *7*(CSCW1), 1-18.

Paine, D., & Lee, C. P. (2014). Producing data, producing software: Developing a radio astronomy research infrastructure. In *2014 IEEE 10th International Conference on e-Science, 1,* 231-238. IEEE.

Ragin, C., & Zaret, D. (1983). Theory and method in comparative research: Two strategies. *Social forces*, *61*(3), 731-754.

Sims, B. H. (2022). Research software engineering: Professionalization, roles, and identity. *Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep*.

Segal, J. (2005). When software engineers met research scientists: A case study. *Empirical Software Engineering*, *10*, 517-536.

Stodden, V., McNutt, M., Bailey, D. H., Deelman, E., Gil, Y., ... & Taufer, M. (2016). Enhancing reproducibility for computational methods. *Science*, *354*(6317), 1240-1241.

Trainer, E. H., Chaihirunkarn, C., Kalyanasundaram, A., & Herbsleb, J. D. (2015, February). From personal tool to community resource: What's the extra work and who will do it? In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing* (pp. 417-430).

Walsham, G. (1995). Interpretive case studies in IS research: nature and method. *European Journal of information systems*, *4*(2), 74-81.

Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., ... & Wilson, P. (2014). Best practices for scientific computing. *PLoS biology*, *12*(1), e100174.