Influence apart from Adoption: How Interaction between Programming and Scientific Practices Shapes Modes of Inquiry in Four Oceanography Teams

Kateryna Kuksenok

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2014-2016

Reading Committee:

Cecilia Aragon, Chair

James Fogarty, Chair

Gina Neff

Program Authorized to Offer Degree:
University of Washington Computer Science and Engineering

University of Washington

Abstract

Influence apart from Adoption: How Interaction between Programming and Scientific Practices Shapes Modes of Inquiry in Four Oceanography Teams

Kateryna Kuksenok

Co-Chairs of the Supervisory Committee:
Cecilia Aragon
Human Centered Design & Engineering

James Fogarty
Computer Science & Engineering

Scientists have been producing and sharing code for decades. Code work done by scientists spans simulation, data processing, analysis, visualization, communication, and data stewardship. Robust instrumentation generates data beyond the scale and comprehension of individuals in their current tools, requiring new approaches to automation and collaboration. Sophisticated web frameworks enable more interactive web portals for displaying data or simulation results to various stakeholders. Educational initiatives that target scientists learning to program are increasingly available, and increasingly reach enrollment limits. Given this context, how is programming in science changing? I argue that changes enacted in scientific programming practices are intended to lead to code that is not only exists and runs for a long time, but continues to be understandable, usable, and extensible. My dissertation examines the interaction between coding practices and scientific inquiry. Although deliberate change is oriented toward this goal, a particular tool or protocol does not require sustained use by a group to have impact on the work practices of that group. In this dissertation, I develop an alternative conceptual framework for reflecting on the goals and outcomes of change.

My findings are based on over 300 hours of observation of a total of 46 scientists from four different oceanography groups. Of the 46 scientists, 21 comprise the core study participants,

doing code work at graduate, post-graduate, and faculty levels. Of the four oceanography groups I studied, two focus on simulation and two on observational data analysis. All engaged in deliberate, reflective change of their programming skills and practices. In collecting and analysing qualitative data, I focused on "code work" in a broader sense, rather than referring to "scripting," "high-performance computing," "scientific software engineering," "data science," or other more specific terms that imply particular working environments and aesthetics. Maintaining an inclusive scope allowed me to not only draw parallels between these practices, but also to consider ways in which they intersect and influence one another. Particular practices or philosophies can be pervasive through all layers of code work, from maintaining a co-authored LaTeX-typeset manuscript in GitHub, to "adding biology" to a well-established model, to implementing an automated test suite for an analytic pipeline that generates daily results and images for a web endpoint.

I propose a conceptual framework of change and use stories from my qualitative study to illustrate its components and dynamics. This framework defines relationships between (1) *the working environment*, which is subject to deliberate change; (2) *the perfect world*, which directs that change; and the (3) *moment of flux*, which constitutes taking action to bring about a change and its immediate outcome. The working environment combines resources that are technical (e.g., iPython Notebook, Google search), cognitive (e.g., looking at many small charts encoding information in a familiar and consistent way to aid quick understanding), and social (e.g., a shared office with frequent "hey, how do you [do a particular tricky thing]?"). The working environment is subject to change, including changes in not only the technical components (e.g., tools) but also cognitive (e.g., skills) and social (e.g., communication practices and language). This change is informed and directed by a collective imagination of a perfect world: the moving target to which possible modifications to the current way of working can be compared. The *moment of flux* when a scientist elects to pursue deliberate change requires both momentum and opportunity, which can arise in

the wake of a breakdown of the prior approach, in the space created by embarking on a new project, or through an energizing workshop or group event. These situations allow the exploration of options that are already in the awareness or intention. Actually making the "leap" of deliberate change to integrate an unfamiliar component or learn a new skill is associated with uncertainty and the possibility of disappointment or failure.

The conceptual framework I propose creates optimistic vocabulary for reflecting upon these changes. As projects involve more people, longer time spans, and more ambitious collaboration between disciplines, understanding how coding practices influence scientific inquiry is increasingly important. The discussion of "best practices" in open science encourages the sharing of negative results and disappointing data as a top priority. This call for reflection on failure must be extended to include code work. With data as well as with code sharing, repeated "best practices" are not sufficient to inspire change, even for those scientists who openly feel they "should" do it. I present qualitative findings that demonstrate concrete ways to deliver interpersonal rewards in the wake of particular efforts not panning out as well as hoped or intended.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This document is essentially a systematic reflection on a variety of things that I learned from dozens of inventive, curious, and reflective people. It would not be possible if these very busy people did not take the time out of their days (over weeks and months) to explain what they were doing, sometimes even to endure my hovering over them with questions all day. I am grateful for their patience, their generosity, and their goodwill towards me.

The University of Washington CSE and HCDE communities have had a massive impact on my personal and intellectual growth. Each person on my reading committee has been extraordinarily supportive, and their insights pushed this work from an open-ended observation to a coherent set of concepts. My co-advisors, Professors James Fogarty and Cecilia Aragon, are two of the most dedicated mentors I have ever had the pleasure to know. Through each of my struggles, despite their own, they held my best interests in mind, as they do with each of their students. Their respective complementary research skills and tastes have challenged and inspired me to develop my own. Professor Gina Neff has gone above and beyond, offering me feedback, guidance, and perspective throughout the analysis and writing process. Many peers and faculty in CSE and HCDE enthusiastically shared with me their time, energy, and insights, in the best possible demonstration of what it is to be collegial.

DEDICATION

To Olga and Sasha, without whom I could not have started this endeavor.

To Heinrich, without whom I could not have followed through.

Chapter 1

INTRODUCTION

The most common, unprompted comments from the scientists that I observed and interviewed in this research were: "I am not a real programmer," "you are about to be really bored," and "not a lot of people do what I do." In the chapters that follow, I characterize **deliberate change** in specialized code work carried out in a context where those who program share little overlapping programming background, and do not typically identify primarily as programmers. In this manuscript, I report on an 18-month qualitative study of four different oceanography groups in the Pacific Northwest. Although the results and discussion are specific to the study context, the implications of the findings may extend to other scientific contexts characterized by a wide range of complementary research methods and adjacent disciplines, and where data and code are routinely used to generate, communicate, and verify scientific findings.

This is a case study of scientific code work in oceanography as a "radically unstandardized" [60] scientific environment. The study of the ocean and its processes spans a variety of disciplines, approaches, questions, applications, geographic areas, and periods of time [36]. Existing data and code sharing practices exist not only in spite of the diversity but also, in certain ways, because of it. Existing computational practice that are effective and substantial enough to have allowed researchers in this diverse field to measure, track, and predict global climate [18]. Researcher groups simultaneously "own" particular methods, which can be completely different from those of other groups. In this way, multiple groups study different geographic and temporal ranges, with different objects of study, at different resolutions work. Their complementary findings are then compared and triangulated into a complex coherent

picture. The major differences between different data sets make the resulting synthesis far richer.

This research adds to the literature on the study of scientific work, particularly on the study of programming practices in science. Software has long played a role in oceanography research, e.g., for high-performance numerical approximation / modeling work, or coupled with instruments for sample analysis and data collection at sea or from sensor networks. Although sharing of code and data are not themselves new to this setting, there is a *zeitgeist of change* expressed by some in the field. This study focuses on those who are engaging with this zeitgeist, on the character of this change. This "zeitgeist" is associated with an openness to influences from software engineering, often summarized by "best practices" and how they can be applied to scientific work. I am interested not only in understanding how these practices are adopted and adapted by scientists, but how this interacts with research in computer science. Furthermore, I am interested in the implications for computer science research that aims to building tools targeted at scientists users. In Chapter 2, I elaborate further on the related work and the scholarly context of this research. In Section 2.1 I review literature that studies scientific software production. In Section 2.2, I provide some readings of designed interventions, including tools and educational workshops. Lastly, in Section 2.3, I offer a short description of how code work is deeply intertwined into oceanography practice, and inseparable from scientific or data work.

During this study, I expected a considerable change in my understanding of the setting. An evolving understanding co-incided with revisions to the research questions (RQs). Below, I include the RQs formulated at the outset of the work in the proposal, along with the subsequent revisions and rationales.

RQ1-2: Meanings and definitions; Change and challenges

> *Initial RQ1:* What is the division between "data science" and the scientific work? What constitutes data science?
>
> *Initial RQ2:* What are the challenges in integrating "data science" practice into various

oceanography work processes?

Revised and combined: What *new* programming skills and practices are being included in "code work" in oceanography? How are these new programming skills and practices integrated into existing scientific practice in oceanography?

Justification: Code work is deeply integrated into data production (e.g., [45] on co-production of code and data), and there are many code and data sharing practices in oceanography that predate the more recent "data science" zeitgeist. The revised question shifts focus away from looking for distinctions, and instead to understanding means of integration.

RQ3: Social context and influences.

Initial: What are the strategies that groups develop for addressing these challenges?

Revised: What role do interpersonal relationships and social processes play in starting and sustaining change?

Justification: The working environment in which code work is embedded is highly personal. Although the social context plays a massive role, and there are some group-level strategies that I report on, my analysis of change mechanisms included interpersonal influences that were not collectively developed or collectively carried out.

RQ4: Implications.

Initial: What are the implications of the strategies on the design of technology and educational initiatives?

Revised: How can a holistic conceptual framework inform evaluation of technology or educational interventions?

Justification: This is still primarily an implications question, adjusted for the revisions of the other questions. A focus on *evaluation*, rather than *design*, arose from the nature and content of the findings: the interviews provide a unique perspective on how people select various available options, how they decide how long to pursue them, and how they reflect on the experience.

The conceptual framework described and justified in Chapter 4, 5, and 6 address revised and combined RQ1-2. Chapter 5 focuses on the "new" of "best practices" adopted and/or adapted from software engineering practice. Research question RQ3, about social influences, is also addressed by these main chapters. Chapter 4 introduces the "collective imagination of the perfect world" as a socially-situated driver for change, and Chapter 6 delves more deeply into the mechanics of an individual deciding to enact change to their working environment, which includes interpersonal relationship that build awareness, momentum, and feedback.

These revisions reflect a gradual grounding of the study in daily activities, and a stripping away of overloaded terminology. Terms like "data science" and "version control" can mean very precise but very different things that vary from person to person, and thereby obscure which of the related implications are brought along. (Making code modular and extensible? Open science? Extensive documentation, or good-enough documentation?) Many disciplines are included in the broader movements of data science, reproducibility, collaboration, and open science movement(s)[1]. Oceanography is a relatively young and wide-ranging field with an extensive history of data and code sharing necessary for the interdisciplinarity and methodological innovation core to its inception and growth [2]. My focus broadened from "data science" to "code work." The details of my methods, settings, and informant population are described in Chapter 3. Aside from ocean scientists[3] who program for at least a part of their time, the study included participants from the Computer Science & Engineering department and the eScience Institute at the University of Washington (UW), but without treating these as distinct communities, rather as collections of individuals playing different roles in their respective teams.

---

[1]Reviewed in Donoho's recent essay on the recent historical context [16]. The section on sensor networks in Borgman's *Big Data, Little Data, No Data* [11] is relevant to issues of oceanographic data collection, and complements the discussion on infrastructure in *A Vast Machine*

[2]Kunzig offers a general history of the field in *Mapping the Deep* [36], and Edwards offers an account of the sociopolitical context of climate modeling in *A Vast Machine*. Both books are used for context-setting in this dissertation.

[3]from Oceanography at UW and Earth, Ocean, and Atmospheric Sciences at UBC

In Chapter 4 to 6, I introduce and illustrate a **conceptual framework** that addresses the revised variants of the research questions, and which is summarized in Figure 1.1. Chapter 4 introduces the *perfect world* as a guiding goal for change in the *working environment*. Chapter 5 shifts attention to ways in which the language of *programming "best practices"* interacts with the collective imagination of the perfect world (and, in turn, the working environment) among the oceanographers. Chapter 6 takes a closer look at deliberate change, which is treated as incremental (relative to the overhaul that attaining the "ideal" would require), but which is riddled with uncertainty that must be coped with.

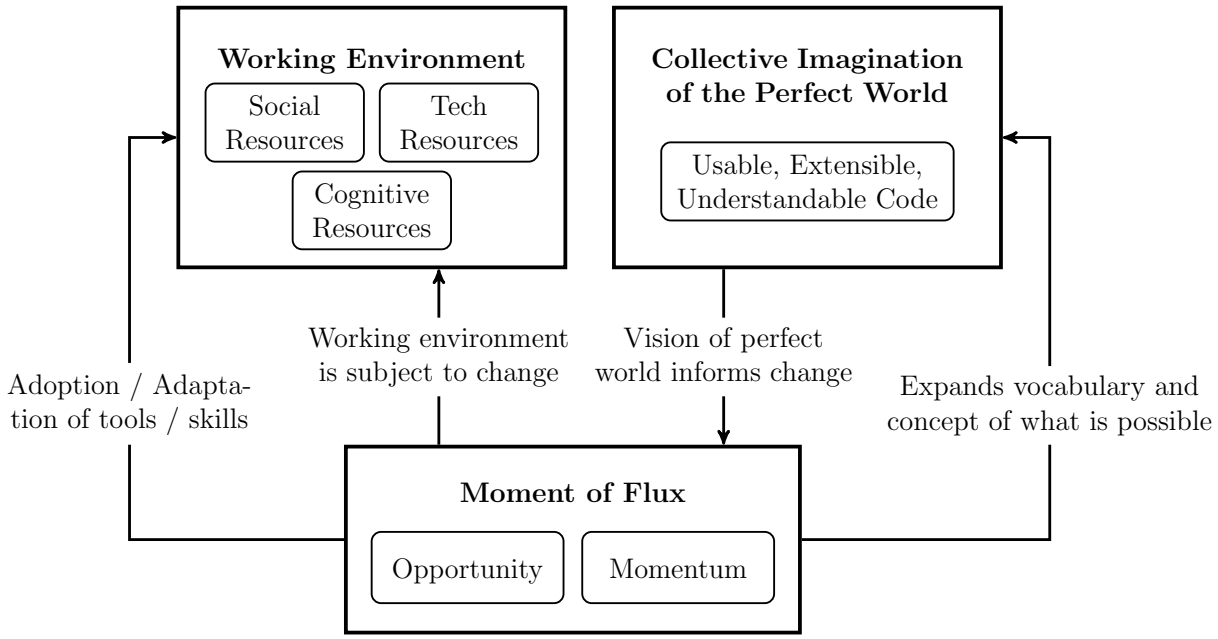

Figure 1.1: The proposed conceptual framework, showing influences that the *working environment* and *the collective imagination of the perfect world* exert and are subject to, relative to *the moment of flux*, that time when the change occurs.

This work contributes an operationalization of the values that are expressed by the informants, and the primary implication is in re-formulating the means for evaluation of interventions in scientific code work. Though intended to be accessible to a wide array of scholars currently engaged in reflective change of scientific code work, these findings are aimed pri-

marily to computer scientists who are actively designing tool or educational interventions. Addressing RQ4, Chapter 7 offer several case studies that (1) demonstrate the use of the conceptual framework set out in Chapter 4-6, and (2) offer alternative ideas of success and means of evaluation, as articulated by study informants.

Chapter 2

BACKGROUND AND RELATED WORK

In this chapter, I review background and related literature about what code in oceanography does and how it is made. In Section 1, I focus on issues relating to production of code in a scientific context. In Section 2, I offer a short summary of the study of the ocean, pointing out various loci of code work.

Studying anything about the ocean is difficult for many reasons. The ocean covers two thirds of the earth's surface. Regional and global patterns of movement are subject to the seasons, as well as major anthropogenic influences, such as ice melting and producing freshwater with a different density than its surroundings. Organisms use up nutrients, die, and reproduce over the course of a day. Human activity, such as overfishing, influences the inhabitants of a complex dynamical system in a manner that is not only hard to interpret in term of future implications, but it is also hard to understand in the present. How many organisms of a particular kind are there? Even if they are big enough to count, they move all the time. What was the visibility underwater when the counting was done? How much of some nutrient or substance is there? Was it measured by submerging an instrument to a certain depth while at sea? How certain are we that the ship, in its movement, was not carrying anything in its wake, or otherwise contaminating the findings? What is the shape of the ocean floor, the overall character of circulation patterns, the changes of salinity or temperature over the seasons and over time? How do you know your observations are reliable? How do you know your model represents something meaningful about the natural world? The study of the ocean is associated with many uncertainties are different levels. Skepticism and a plurality of complementary methods help make this uncertainty productive, rather than an insurmountable barrier.

Code work is part of this plurality of methods, integrated differently in different ocean science contexts. The place of software in this multi-disciplinary field has been both to enable computational modeling and to enable generating, handling, curating, and comparing increasing quantities of data. Figure 2.1 illustrates a simplification of the loci of "code work" in oceanography. Computational modeling allows scientists to make causal claims and formulate hypotheses which can be tested using observation. More robust use of database systems allows the possibility of integrating observational data. Greater scale, resolution, and integration of observational data strengthens its ability to be used as evidence in increasingly complex claims about how the ocean's biology, chemistry, and physics fit together into a coherent system. The coherent understanding of this world benefits from data stewardship efforts that make data more robustly available to scientists who would not otherwise have personal access to that data.

2.1 Study Focus: Code Products and Code Work

Borgman describes data as the "glue" of collaboration between scientists and technologists [12]. Indeed, open data is a crucial means to integrate and accumulate knowledge [52] in a multidisciplinary environment, particularly one that is "radically unstandardized" [60]. From the software engineering research side, particularly when it comes to numerical modelling traditions in the sciences, there is a "lack of separation of concerns,"[1] no sense of abstraction, where "the problem to be solved entirely [is] mixed with numerical approaches and target-dependent information" [46]. Heaton and Carver note a similar claim in their systematic review of claims about software engineering practices in the sciences, specifically modeling [27]. Sletholt, Hannay et al also find that "the definition of test cases for validation and verification of the software is perceived as challenging ... it is often not obvious to stip-

[1]Patel et al, in a study of programmers attempting to implement machine learning applications and having difficulty because they focus too much on algorithm selection and neglect data debugging [47]. This can be summarized as the opposite of the indictment that the scientists worry too much about their data and not enough about their code, leaving us with the unremarkable hypothesis that in problem-solving, people focus on what they know.

ulate whether an error lies within the scientific theory or in the implementation (numerical approximation) of that very theory" [58]. In this sense, the software can also be described as "glue" - once applied, inextricably and irreversibly linked to the other components. Or, as Ragan et al note in a Nature article on programming environments for microbial ecology: "Although computation can be a language that bridges many disciplines, additional 'glue' is often needed to make the requirements mutually comprehensible to diverse members of a project team" [51]. Here, the "glue" is more the kind of "social glue" that Henderson identifies in her work on engineering sketches, where physical, flexible representations which are preferable to the more rigid technological environments because they act, by virtue of their flexibility, as a "social glue" between different communities of practice [28].

The "glue" that holds this section together, in its wide-ranging review of half a dozen fields of scholarship, is the existence of scientists who take part in, or want to take part in, many different kinds of code work in the course of scientific craft. The study of adoption of tools is then linked to programming skill acquisition, which is in turn influenced by the context of the scientist and the extent to which certain kinds of code work are pervasive or rewarded. In the study of large collaborations (e.g., oceanography [60] and radio astronomy [45]) or other contexts where there is relatively clear separation of skills and interests (e.g., data gathering at sea with robots [12]), communication and collaboration practices are the focus of study. Outside of these massive projects, the principal investigators (PIs) are responsible for managing diverse groups with disparate skill-sets and backgrounds, for determining which innovations to invest (time, money, and energy) in, for creating additional roles within the group, and for advising the early-career members of that group not only on success within it but also beyond. This may be especially salient in disciplines like oceanography, where there are many different research methods and many different levels of associated collaborative work or major expenses, including software licenses, laboratory equipment, staff roles.

2.1.1 Code Starts and Ends with Data

A large body of work in computer science and adjacent fields (from information visualization to software engineering to computer-supported cooperative work) concerns itself with how scientists work with data as the fundamental building-block of empirical study of the natural world. In this section, I highlight scholarship that identifies ways in which code is deeply integrated with the production of data, as well as the dissemination of data in this study context.

In *A Vast Machine*, Edwards writes: "It is easy to forget that data are never an abstraction, never just "out there." We speak of "collecting" data, as if they were apples or clams, but in fact we literally *make* data" [18]. This quote is excerpted from a passage that describes how the act of digitizing an analog temperature reading is an act of constructing data. Data are *co-produced* with code in science, as Paine and Lee find in an ethnographic study of a radio astronomy collaboration [45]. As environmental data is increasingly more available, instances of "data avalanche" in the 2000s [29] or "explosion of data" in the 1950s [18] correspond to vocational shifts in what it means to "be" an ecologist or "do" ecology [31].

Paine and Lee additionally write that the *legitimacy* and *importance* of code work among scientists who "acknowledge that this work is necessary and a fact of their day-to-day lives, yet still do not entirely consider such work to truly be the 'scientific' work in and of itself." Paine and Lee stress that "iterative system development is key for answering their scientific questions and the development of the necessary research infrastructures" and therefore that "such 'non-scientific' work has a deep and lasting impact on the group's scientific outcomes."

Baker et al introduce *Ocean Informatics*, an area that inhabits intersection of oceanography, information sciences, and social sciences, drawing on the lessons learned from two independent programs with some common data management practices from over a decade [4]. Below, I summarize the most salient points of the timelines of two programs from Baker et al's manuscript: Palmer Station and JGOFS.

In 1990, the program Palmer Station was established as a long-term ecological research (LTER) site, and the data was made available online in the form of static text. Given that the history of triangulating data and theory reaches back almost a century ([36]) and the history of computational modeling over half a century ([18]), the formalization of data access through an online endpoint is relatively recent. Nevertheless, it still predates Apache Subversion (founded 2000) and GitHub (founded 2008); it also predates the initial work on the iPython Notebook [49], which began in 2001 although the stable release came over a decade later. Aside from the point of precedent, I also want to highlight that because of the need for comparisons over time spans, time scales, and between geographical regions, long-term research sites are especially useful. This also means that data is available now about the era following the 1990s at scale and resolution that did not exist before that time, or exists in inaccessible format. The stewardship of *historical* data is also important, not just forward-looking data infrastructures.

Over the next decade, the capabilities of JGOFS illustrate that a data endpoint can be arbitrarily complex and feature-rich:

> From its beginning in 1994, JGOFS had a "Data Management Office" with technical staff that worked together with investigators; "much of the collaboration focused on issues related to quality control and the collection and subsequent publication of complete metadata for contributed data sets." "All process study data were ingested into an object-oriented, relational database and made available via the World Wide Web. Using a standard Web browser client, users of the JGOFS data system can generate custom data sets that match their research interests by combining multiple data sources 'on-the-fly'." In early 2000s, "to meet requests for data queriability and requirements for networking," new system was designed featuring "online data access, strategic integration and visualization." "Data and metadata management is offered through web interfaces with tiered permissions that enable data provider participation in making their data accessible. The new

> system is built upon a relational database with an object-oriented API layer that supports Web-based data query." [4]

The comparison of findings and predictions includes model output, which is addressed by the solution in [29] and ocean data view [56], both of which focus on readjusting the grid and thereby enabling coherence between disparate data sources [24]. Enke et al, writing about barriers to biodiversity data sharing, also note the importance of combining observational and model data in data access endpoints [19]. Baker et al write: "As the JGOFS program transitioned from process-oriented field studies to modeling, the data system was extended... Synthesis and model results, larger in volume and often global [rather than regional] in scope... required a more graphically oriented user interface and extended visualization capabilities" and the technical staff "worked closely with investigators to provide timely availability of data during the active research phase and to ensure preservation of the completed data collection as an important part of the [program] legacy" [4].

2.1.2 Is code in scientific contexts special?

In the previous section, I make the argument that code work is an important facet of scientific work that deserves study, particularly study that looks at code work as deeply integrated into other activities. In this section, I ask, what is distinct about software work in science relative to other contexts? Sletholt et al write: "in contrast to the development of, say, administrative or business enterprise software, the writers of scientific software cannot determine what the correct output of an application should be in the traditional sense. Also, the software may evolve through the combined effort of a number of scientists over the course of many years, continuously adding new functionality to the system [58]." This strand of research characterizes software production in the sciences as qualitatively different from other contexts because of a more uncertain concept of the end result or outcome [27]; requirements are volatile [58]; subject to continuous maintenance [58], following through with which is primarily socially-motivated (in a philosophical at least as much as a transactional sense) [62];

not subject to formal design process [27, 45] but nevertheless adapting various programming best-practices, most notably elements of the agile process [27, 58]. In some cases, particularly in numerical approximation code, scientific code can exist without appropriate maintenance, relative to the high-performance computing hardware on which it is meant to be run [46]. Heaton and Carver provide a systematic overview of studies of software engineering practices in scientific code (again, numerical approximation / modeling in particular), concluding also on the general lack of maintainability or readability of code [27].

Howison and Herbsleb note that "Unlike other technologies supporting science, software can be copied and distributed at essentially no cost, potentially opening the door to unprecedented levels of sharing and collaborative innovation. Yet we do not have a clear picture of how software development for science fits into the day-to-day practice of science, or how well the methods and incentives of its production facilitate realization of this potential" [30]. This articulation of software practice as offering unprecedented potential is a crucial part of the vision of software in the perfect world as not only solving problems but fundamentally challenging the daily reality of work. Speaking to code in ecology in particular, Mislan et al note that "even code that is rough and difficult to run on other systems (owing to software dependencies and differences in computing platforms) still provides valuable information as part of detailed documentation of the analyses" [44]. The authors suggest some venues for publishing code, presenting different options side-by-side in a table "Comparison of Common Resources," and urge journals to make code citation more prominent to address this issue. Trainer et al introduce the concept of "extra work," referring to all the maintenance and support work that is not formally accommodated in scientific practice but which is practically necessary to make that code viable [62]. In their typology of motivations for doing said "extra work," every category includes a social connection, an actual interpersonal request, or some other opportunity for recognition of effort or a sense that someone is actually using something. Enke et al, writing about biodiversity *data* sharing, divide all measures to encourage more sharing into "stick" and "carrot" measures, with many more specific im-

plications on the basis of more than 700 survey responses [19], but adding to the chorus of needing recognition for doing thankless and crucial heavy lifting.

Free/Libre and Open Source Software (F/LOSS) is another form of software work that distinguishes itself from the "typical" industrial software setting [38]. In a study of software ownership ("a general term used to describe whether one person has responsibility for a software component, or if there is no one clearly responsible developer"), Bird et al demonstrate and explore the claim that "when there is no clear point of contact and the contributions to a software component are spread across many developers, there is an increased chance of communication breakdowns, misaligned goals, inconsistent interfaces and semantics, all leading to lower quality" [10]. They note that "new methods ... profess collective code ownership but there has been little empirical evidence or backing of this data on reasonably mature/complex or large systems." Though not speaking of F/LOSS as a whole, they note one such paradigm of software practice which shares some practical components with the F/LOSS ethos described above and makes similar quality claims [10]. Although F/LOSS distinguishes itself from the hierarchical industrial software setting, and vice versa, Leach comments also points out a surprisingly-pronounced hierarchy, with authority "often enforced in displays of aggressive argument and belittling of others." Complementing its community-centric philosophy, F/LOSS is associated with a sense of individual craftsmanship, where "the craft in question is satisfying only when it is seen in terms of a model of knowledge that locates agency and activity within the technologist, who thereby himself pushes back the frontier of knowledge." In this sense, "morality and ethics are inseparable from the objects which the community produces. Politics, and an imagined future, are pursued through the construction and development of software, of objects, which themselves are to carry the burden of, and are believed to instantiate, an ethical and moral vision. Politics in this case can be engaged though writing software in a particular manner." [38]

In their synthesis of literature on the challenges of software, especially communication, Bird et al draw a link between expertise and ownership: those who are responsible for the code have an expertise that is not merely helpful but necessary for future development on a

particular component [10]. On a similar note, Kelly writes that "the scientist is an integral part of the software system and cannot be excluded from its consideration" [32]. Looking via interviews at the perspective of computer scientists engaged at middle-ware development, Lee et al [40] "examines how technologists develop and sustain middle-ware applications over time by leveraging expertise and partnering with different research domains in order to achieve log-term infrastructural goals. Activities to "sustain their software development research agenda well beyond the lifetime of project Lee et al have also emphasized the intentional personal networks as key - the human infrastructures that sustain cyber-infrastructure projects [41].

There are different ways to bound scientific software, relative to other kinds of software work. The place of software has been broadly categorized as "including data analysis, simulations, and managing workflows" [30], and as deeply integrated with scientific pursuit in [58]. Kelly bounds her study of scientific software production as extending to "application software that includes a large component of knowledge from the scientific application domain and is used to increase the knowledge of science for the purpose of solving real-world problems," and excludes "software written to become a commodity product ... to verify [safety procedures] [and] to control equipment" which "has to be correct, to the exclusion of all else" [32]. In addition to coding per se, we may also include end-user programming using complex systems that present the scientific community with an astounding array of possibilities, all of which demand skill-acquisition and attention. LabVIEW (see [66]) and Excel are commonly-used "tools" but also "end-user programming" environments, with barriers to learning not unlike those of applying programming skills, such as design and difficulty in understanding and explaining unexpected behavior [34]. Tools can also be intended to support learning: Cervantes et al have considered ways in which skill-based engineering expertise may be shared in the eScience context [13], such as via wikis.

Kelly points out that scientists identify their coding practices largely by non-adherence to a particular software production methodology [32]. This "amethodical" narrative is presented in contrast to the methodical one by Truex et al [64]. The methodical, privileged approach to code work holds that software systems development is more controlled and rational than

it is in lived practice of industry software engineering. Truex et al create an alternate narrative that "occurs in completely unique and idiosyncratic forms". In the lived reality of coding, neither exists in its elementary state, but the amethodical narrative is marginalized in reflection about software engineering practices. Building on this work, Philip et al study amethodical remixing, or the use of copy and paste and snippets [50]. Though these practices are not included in textbook, privileged accounts of development, they are nevertheless present. Implications offer design of tools that support more realistic everyday scenarios of re-mixing, which blend the dialectic methodical and amethodical narratives.

2.1.3 Best Practices: The Universal Necessity of Sharing

In this section, I pull together discussions on programming best practices, as well as meta-discussion on the matters of the practical and the beautiful in code work. In synthesizing both of these bodies of work, I distinguish code products from code practice. Additionally, I focus on sharing and making shareable as the two main values.

The term "best practices" refers to industry-wide, shared ideas about the best way to do things. These often stress putting code (and data) online for others to access [61], even if it cannot run but at least as a documentation of scientific activity [44]. Wilson et al offer 8 concrete directions, including things like "plan for mistakes (turn bugs into test cases)" which connect a minor expected roadblock or problem (a bug) into the opportunity for a "best practice" act (testing) [70]. The suggested practices from Baxter et al are a bit more high-level: "1) design the project up-front; 2) document programs and key processes; 3) apply quality control; 4) use data standards where possible; and 5) incorporate project management" [7].

In addition to the software-related strategies and practices above, additional practices concern making data more effectively available through "deliberate documentation" and persistent quality access [4]. The list of best practices is in constant flux, as noted by Sletholt et al who conduct a review of agile practices reported in scientific publications, and

who characterize scientific practice as distinct from other contexts in that it is relatively more ad-hoc and grounded in experience [58].

In his reflection on the design and implementation of the Software Carpentry curriculum for well over a decade, Wilson writes that computational scientists do not have the publishing incentives to prioritize code quality, so the course does not attempt to "'sell' quality directly... Instead, it starts from the fact that the only way to improve productivity is to improve quality" [69]. The claim of the "fact" is an illustration of the relationship between moral and material aspects of code work:

> A central component of the F/LOSS ethos is that working openly and sharing the source code of software enables improvements to evolve more effectively, and that as a whole "better" software is produced. The concept of 'better software' (a material judgement), which arises from 'better' processes of production (a moral judgement) conceals another complex series of understandings and judgements generated by familiarity with and proximity to the workings of the machine (computer) itself ... the beauty of code comes to be an aim in itself in this overlap of practical and moral truth. [38]

Leach reports on an ethnographic study of free/libre and open-source software (F/LOSS), going on to point out how his informants would often claims that the "messiness" of proprietary software is the primary reason it remains inaccessible, and that its producers "would dare not to open source their code because [of shame over] how its functionality was achieved." The informants quoted in my dissertation often expressed hesitance to open up source code on account of its messiness, which is distinct from the concern over attribution as articulated by Enke [19]. "I am not a real programmer" and "it works, but it's not the most elegant way to do things" were common asides among the participants of this study when describing their own work. The craftsmanship of code, as elevated by the F/LOSS ethos, trickles into the self-reflective scientific code discourse and touches on "open science" and related notions. For example, Software Carpentry has engaged in the F/LOSS

community and value system since its inception in the 90s [69]. The interpretations of "best practices" for scientific coding invariably stress sharing in general and sometimes open source in particular [44, 61].

The mechanics are distinct between these notions, but they all embody a common value of sharing, which is simultaneously powerful enough to warrant a great deal of discussion and non-specific enough for that discussion to mostly focus on critiquing shortcomings. Howison and Herbsleb identify several different *software production systems* articulated in terms of the social and institutional contexts and in incentives [30]. One these software production systems is *parallel software practice*, which is similar to what Paine and Lee focus on in [45], and which Howison and Herbsleb do not critique. On the other hand, the authors suggest that a "dual science and software practice is a somewhat questionable proposition for working scientists" because aside from creation effort, the maintenance effort is unreasonable and inescapable. This maintenance effort is the "extra work" that Trainer et al identify as motivate primarily by social means [62]. For example, direct interpersonal exchange over email or with a past colleague can help prioritise an endless list of possible features with unclear ramifications for scientific users. The discussion of incentives and motivations concerns itself with the *material*, while the discussion of increasing sharing or code "messiness" concerns itself with the *moral*. Leach notes that the material goodness of better software is achieved by the moral goodness of better practice. The assumption behind critiquing and problematizing incentive structures reverses the relationship, making the moral goodness of open-science-related concepts accessible through the material goodness of rewarding effective code work practices.

2.1.4 Coping with Code

Vocational shifts in the amount of computational work affect different areas and fields differently. The coding practice of a computational modeling researcher can come under critique for lacking abstraction or formal testing [27]. The researcher who works with increasingly large amount of field data that demand learning new ways of automation is urged not only

to learn and use these even-evolving skills, but also share the results as an increasingly important record of scientific work [44]. The broadly-scoped values and philosophies of sharing, automation, and scalability are in practice implemented by way of "small-scale and immediately practical issues," which is why one of the most popular workshops for scientists learning programming calls itself Software *Carpentry* rather than Software *Engineering* [68]. Section 2.1 reviews literature that investigates, critiques, or seeks to change *software practice in scientific contexts*. Section 2.2 subsequently expands to include *designed interventions* that seek to transform scientific practice, spanning both tools (e.g., visual analytics pipelines) and social initiatives (e.g., Software Carpentry workshops).

Technological capabilities change over time, particularly in observational oceanography, and those affected undertake a variety of strategies to cope with change, including learning [60]. On the basis of an ethnographic study of a large oceanography data infrastructure collaboration, Steinhardt and Jackson introduce the notions of "anticipation work," which includes "the mundane, local, and sometimes highly personal accommodations to the future that always accompany the more formalized and/or speculative dimensions." One of the examples of these "highly personal accommodations to the future" was the intention to learn new skills in anticipation of the next career move after the infrastructure project had wrapped up. With time as an analytic lens in an ethnographic study, Chen et al point out that the demands on human time to learn the necessary skills and to use them to make a multi-component system behave as desired are a notable conflict in the lifecycle of a scientist's use of a high-performance computing (HPC) system [14]. Nevertheless an overwhelming majority of programming work done by scientists is not with large systems, as Hannay et al find [26]. Regarding databases, Franklin et al argue for co-existence over integration, where "semantic integration evolves over time and only where needed" because "the most scarce resource available for semantic integration is human attention" [20]; though the existence of integrated databases would present an improvement over the inefficient manual data-retrieval and transformation currently in place among many scientists [19].

In a recent summary of the adaptations necessary "in the face of changing science" [9], Bietz and Lee highlight the tension of creating contextualized, tailored software solutions against the maintenance and design of a long-lived cyber-infrastructure. In a similar fashion that foregrounds software construction and data use in the study of scientific collaborations, Ribes and Finholt identify as a major tension in cyber-infrastructure development between "respecting current work practices" and "transforming scientific practice" [54]. Endeavors defined by their ambition to transform scientific practice into a more collaborative endeavor able to make use of orders of magnitude more data are therefore not only risky technological undertakings, but are also associated with tensions across values and meanings between communities involved. In considering the failed as well as the successful infrastructures within the eScience umbrella, Zimmerman points out a gap between "how systems work and how users expect them to work," and, based on interviews of ecologists, notes that their "collection of data for re-use mirrors the standards that guide the gathering of their own data in the field or laboratory" [75].

2.1.5 Visual Meaning-Making

Gilbert makes the case for visualization as a crucial metacognitive skill in science and, inextricably, science education [22]. Grochow et al introduce a hybrid client-and-cloud analysis and visualization environment to support the many kinds of comparisons and transformations necessary [24]. Their system is based on Trident, and includes improvements in efficiency [5]. ROMSTOOLS also offers an "integrated toolbox" with some global datasets (such as SeaWIFS, a global satellite image which uses color to estimate surface biomass over the recent decades) and MATLAB programs for the ROMS ocean simulations, noting that "tools for visualization, animations and diagnostics are also provided" [48]. Other tools include visualization environments (e.g., ocean data view [56], Trident [5], COVE [23], and COOS [42]), data management approaches for the specific challenges of data collection at sea [8], programming environments to support the writing of better software (e.g., [46]). Interventions aimed to augment or transform various different parts of the existing scientific process much

contend with an even broader interpretation of "tool," for example noting that "the success of paper field notebooks can be attributed to their resiliency to damage in rugged terrain – a sheet of paper torn in half becomes two; a tablet computer torn in half does not beget two computers" [73]. These as much as the products of interactive visualizations constitute a record of transformation in the course of scientific work [37]. In an ethnographic work of the team studying Mars via the Mars *Rover* and the images it produces, Vertesi articulates "*drawing as*" as an "analytical frame" with which "instead of asking what is drawn, asking how and what it is *drawn as*," and which, thereby:

> ....requires the analyst to inquire into the work involved in crafting an image that can be taken up in practice as a transparent representation of the object in question. This bypasses questions of reference – how the image is tied to an object in the external world or whether the depiction reveals the object's essential nature – to reveal how image-making in science inscribes a scientific community's values, organization of work, or epistemology onto the object at hand [65]

2.1.6 Summary

Software also plays a role in collecting, handling, analyzing, displaying, and sharing observational or field data. As an example of software as a component of a more larger data collection scheme, consider the following abstract excerpted from a 1996 publication by researchers from Woods Hole:

> Traditional methods for determining spatial distributions of planktonic taxa involve net, pump, and bottle collections followed by the tedious and time-consuming task of plankton sample analysis. Thus, plankton ecologists often require months or even years to process samples from a single study. In this paper, we present a method that allows rapid visualization of the distribution of planktonic taxa while at sea. ... We describe the techniques used in imaging the plankton, analyzing the video, and visualizing the data. We present an example of at-sea data analysis conducted

> aboard [research vessel at specific place on a specific month] and visualizations of the 3-dimensional distribution of selected planktonic taxa in a 2 2 km 90 m volume of seawater. A video of the image processing and visualization is included on the CD-ROM accompanying this volume and is an essential part of this paper. [15]

This paper has, early on, a section on "system requirements" for the "fully automated at-sea analysis of plankton size and taxonomic composition" has three sections: "(1) High-quality images of individual plankton ... (2) An image processing and pattern recognition system... [including] (a) a software/hardware system ... for preprocessing of the images (including realtime image capture, object detection, focus detection, and digital storage of the region-of-interest); (b) pattern recognition algorithms ... for automated identification and classification of planktonic taxa; (c) the pattern recognition algorithms must be transferred to high-performance image analysis hardware in order to achieve real-time processing capability; (3) a dam processing and visualization system must be developed to analyze and display the distributional data at sea in real-time" [15].

The requirements are all centered around enabling **real-time** analysis that is **not otherwise possible**. Through the rest of the paper, they describe an implementation that actually includes a human intervention component standing in for something that would, in a possible future, be automated. The real-time processing is enabled by any means necessary: a trained researcher on board, specialized hardware for better performance, or whatever is both feasible and enables operation at the viable pace. The paper describes an "interim solution:" a "'point-and-click' user interface for analyzing [regions of interest] obtained at sea and from archived video tapes." the authors also go into detail regarding the "underwater video system," in terms of its specifications and construction, references to a prior publication, and the contexts in which it has been deployed.

I chose this paper as an anchor because it both describes a tightly-coupled software-hardware-user system, but also because it provides more concrete context to the discussion

of code in science, and code in oceanography in particular. The following section delves into this more deeply.

2.2 Study Setting: Oceanography

In this section, I transition to describing the setting of this study: oceanography, and more specifically, code work as it interacts with other ways of knowing in this multi-disciplinary field. In the previous section, I included various discussion of code practice and products. Talking about these concepts in the abstract may under-emphasize just how much the pursuit of science is driven by awe, and the extent to which oceanographers often see software work as arguably the least riveting component of their work, but also an amazing opportunity to build new forms of knowledge. For some, it is at times the least enjoyable, though there are many who derive joy from the particular attentive problem-solving of code work. In this, as in many respects, oceanographers form a diverse and excited crowd. The goal of this section is to provide examples and grounding of what "doing oceanography," practically, involves, and what motivates those who do it.

2.2.1 Identities of an Oceanographer

In an excellent history of oceanography, focusing on the last century to the present, *Mapping The Deep*, Robert Kunzig offers the following description of a British researcher who was instrumental in developing the field's understanding of deep ocean currents (emphasis added):

> John Swallow was **an oceanographer's oceanographer**: a man with a pure love of studying the sea. ... [He recalled of his experiences being drafted into the Royal Navy in WWII:] "I found I was quite happy going to sea and crawling around a ship mending electronic equipment." ... Swallow was a soft-spoken man, modest about his own insights into questions of great import ... a man who was quite happy not to be working on anything "important;' at all; a man who liked quiet and liked

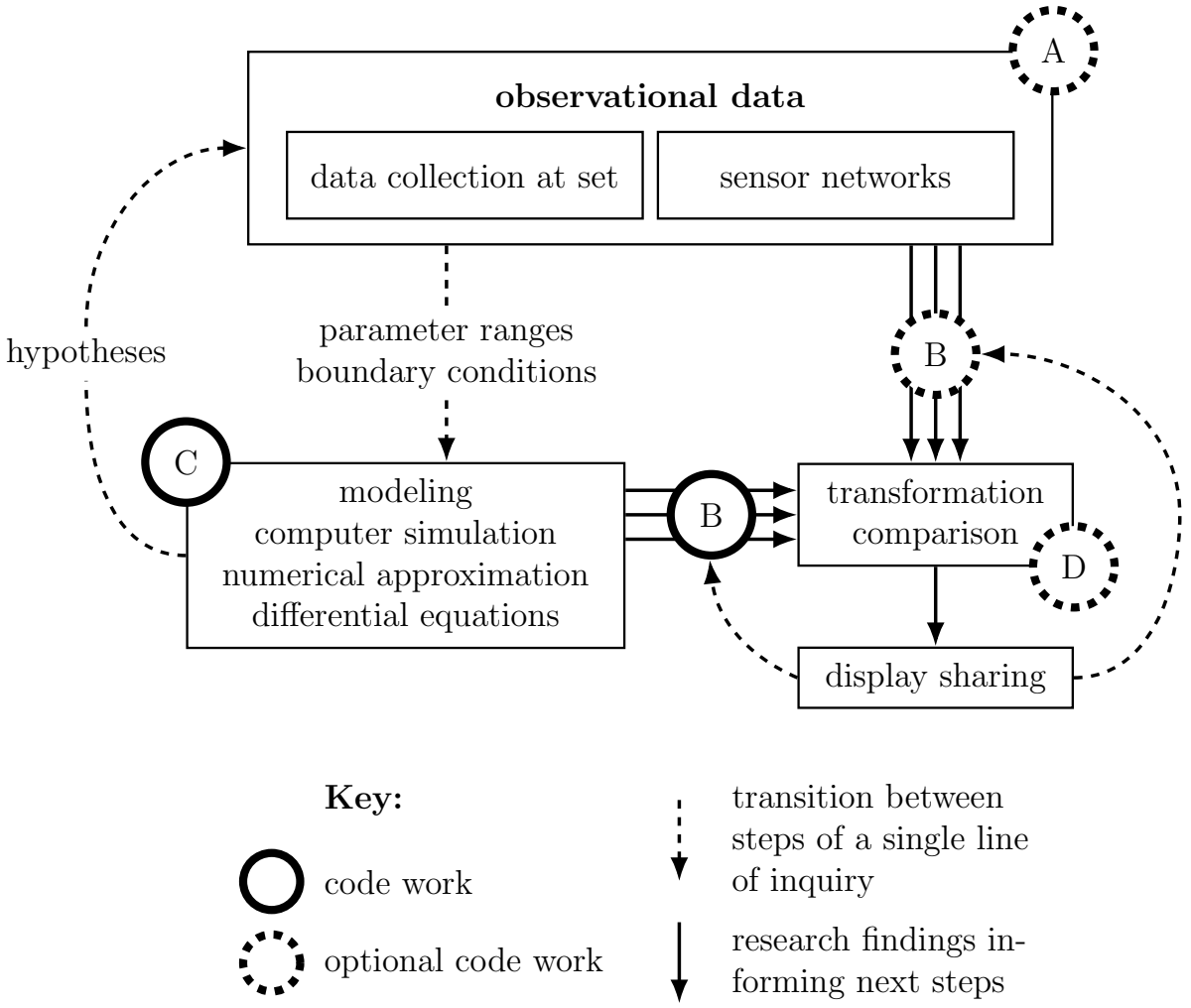

Figure 2.1: **The place of code work in oceanography.** Loci for code work include: **(A)** data processing tightly coupled to collection method (e.g., with code work [15] and without code work [72]); **(B)** data presentation and stewardship; **(C)** existing modeling practice (see *A Vast Machine* on the history of modeling [18], and Section 2.1 for recent studies on software practice in the modeling context, notably the Heaton and Carver review [27]); **(D)** an opportunity for innovation for better comparisons. In this diagram, I distinguish observational and modeling oceanography (described in Section 2.2). Additionally, the dashed border to denote 'optional' code work indicates code work loci with particularly wide ranges of automation, from end-user programming (with Excel or LabVIEW), to scripting (with R or Python), to engineering a custom analytic pipeline. See also 7.1 for a breakdown of where computer science can intervene on the basis of collaboration loci, which complements this more domain-science-focused illustration.

> tinkering with instruments - and who had a great gift for making them work at sea.) [36, pp. 283-286]

Swallow was a physical oceanographer in the late 1950s who sought to collect data to measure deep-ocean currents. This predated satellite navigation, so his floats were constructed to emit sounds that could be picked up by hydrophones within a few miles. Multiple such measurements allowed him to find the position relative to the ship, while the ship itself could fix its own absolute position using a nearby anchored buoy. These observations confirmed theories of intense currents on the ocean floor, capable of transporting nutrients and organisms globally. Later, he decided to release clusters of floats, at depths up to 13k feet (almost 4k meters), and return to locate them every few days or a week. He assumed they would be moving no more than half a mile a day, and as a whole they would show an overall northward flow. However, he was surprised to discover that they were moving much faster (up to half a mile an hour). Furthermore, they did not move consistently northward as expected, but in a surprising variety of direction. This was the discovery of "mesoscale" eddies, "long-lasting but not permanent, with diameters of roughly 10 to 100 miles ... the physical analogue of local weather systems in the atmosphere," but in the ocean [36]!

This story from the late-1950s chapter of oceanography research illustrates (1) the ocean is mysterious and even the fundamentals, like the overall movement of the water, are profoundly difficult to measure, thereby requiring (2) invention and discovery within the discipline that spans theory, hardware, and collective data efforts. For such historical contexts, I include excerpts from Kunzig's *Mapping the Deep*, as well as from Edward's *A Vast Machine* [18]. These are excellent books on, respectively, the recent history of oceanography in general and the history and politics of climate modeling.

What oceanographers consider legitimate activities in their field varied widely. Although all study participants were in a university department encompassing ocean science, none would self-identify to their peers as an "oceanographer" without further explanation. Furthermore, certaintly none would claim the kind of confidence ascribed to Swallow in the

above excerpt. In the rest of section, my aim is to provide enough detail in which to ground the various self-identifications, and the resulting motivations and work environments, that provide the vital backdrop for the upcoming discussion of code work in the next chapters. This background for code work in oceanography is illustrated using participant quotes.

2.2.2 General and Specific Scientific Work

Over lunch, Erin and Lindsay, two post-doctoral fellows,[2] explain to me that earth and ocean sciences encourage a degree of generalization, where people are familiar with every step of the research process from the murky deep to the lab to the spreadsheet. In their case, researchers are expected to be familiar with steps from mathematical representations to numerical approximation methods for solving systems of differential equations to the website with the visualizations. Despite an expectation of competence and understanding at every step, oceanographers do not tend to self-identify as general-purpose oceanographers with no other modifiers attached.

A post-doctoral researcher,[3] Mallory, tells me that there are two kinds of people in oceanography: methods-oriented people and question-oriented people. She explains that it is easier, and "more efficient," to be the former, though she identifies more with the latter group. A researcher can have a method and apply it to different questions - sitting in her chair, she points at the imaginary questions surrounding her; she is the essence of the method, sitting in the center of its observable universe. In this way, she tells me that people get known for their methods: "oh that's the [method/approach for analysis] guy, he'll do [very specific kind of] analysis for you." The alternative is asking a set of questions from various angles: she interlaces her fingers, and forms an oblique mysterious object at which she, the researcher, peers it from different sides. She identifies as the latter kind, which she sees as relatively more rare, having transitioned from modeling and physical oceanography to observational work and working with environmental data focused more on questions of

[2]In the RegionalNowcast-Model team, as introduced in Chapter 3.

[3]In the CustomInstrument-Lab team, as introduced in Chapter 3.

biology. Even in physical oceanography, she had worked on incorporating biology into the model, so the subject of study has persisted through the transition.

2.2.3 Computational Models

Throughout this manuscript, a *model* refers a piece of software that uses numerical approximation approaches to solve systems of differential equations. Depending on the context of the conversation, the *system of differential equations* itself can be referred to as "the model," as can the programmatic *implementation* of some numerical approximation for solving the equations, or a specific *instantiation* (parameterization) of the code (as in, in comparing results of different runs). The equations describe *change* - the movement of particles or tracers from place to place, or their transformation over time. Tracers refer to relevant measurable data, like salinity and temperature.

The domains of interest are bounded by both the surrounding land topography and underwater topography. The latter may be referred to as *bathymetry*, which is the study of underwater topography. For example, the depth included in an analysis or a visualization can come from one of several possible 'bathymetry files.' Sometimes 'the bathymetry' may be described as not quite accurate relative to the real world, but preferable because using it produces more realisti results in the model. These lines that separate water from not-water in the world of modeling are precise, but mediated by the complexity of the model and the state of the best understanding of the world. The parameters and the relationships between measurable elements are also precise within the model, but related to reality by a series of decision made by a group of people or - often - a single individual tacking a specific question. Veronica, a graduate student[4] comments on a model that their group (along with many other groups) use, which has a dizzying multitude of parameters. All the while, she tells me people are working on new packages; during the course of this project, the group had undergone a somewhat involved process of updating the code to an upgraded version with

[4]In the RegionalNowcast-Model group, as introduced in Chapter 3.

some substantial changes[5]. People write packages, Veronica tells me, because "if someone wants to see internal waves, they need a different mixing parameterization," and the model is big and popular enough that there are dedicated developers to make sure everything is compatible.

Models represent "different worlds;" the things that happen in models "are real," they just may or may not match the observed world in one or more measurable ways. The processes of explicit and implicit comparison to observed reality, and the state of the art of understanding reality, guide the decisions of which instantiation of a model is *better*. The model output, therefore, *is never interpreted in isolation* but only in its relationship to other comparable phenomena. Comparability is, therefore, both central and a major challenge, which I will return to again and again throughout the dissertation.

The practice of modeling hinges on comparability. However, comparability is relatively difficult to achieve because of differences between models and even bigger differences between models and observations. Models are different in their spatial domain and the time range they span, as well as the space and time resolution. A model that covers a lot of ground and a lot of time and very high resolution can be prohibitive in its demand of computational resources; these models are shared by oceanographers and it can be difficult to get time to run experiments. Modeling more constrained domains at lower resolutions can be accomplished on individual researchers' laptops in a reasonable time span for experiments and iterations.

What kind of model something is depends on which first-order mechanisms it includes. The atmosphere is the wind at the surface; it is possible to "add the atmosphere to the model," which means including an equation that describes the relationship between the wind and the measurable variables at the surface. Alternatively, one can choose to "leave out the atmosphere" or to use "atmospheric forcings," which means including the influence of atmospheric variables - like the wind - with constants that affect the ocean but without modeling the variable itself. The model is complex. The "physics" of the model describe

[5]intended to make it more usable but actually inspiring the ire of "software advisor" of the RegionalNowcast-Model group; this role, and "The Upgrade" story, are both introduced in Chapter 3.

how movement is handled, but the mechanism may vary in detail, complexity, and realism. What "kind" of a model something is may depend also on the focus of study.

For example, a biogeochemical model is used to study something like CO_2, or a bio model to investigate phytoplankton growth. The focus dictates what first order mechanisms are necessary to include (such as nutrient availability and grazers at at top and bottom). Because a simpler model is a better model - both conceptually and computationally - those mechanisms that do not affect the phenomenon of interest would be left out. Elements like nutrients can be considered chemicals, but an oceanographer wouldn't call it a biogeochemical model unless she has more of a focus on the chemistry, like particle dynamics. Because of this, when a modeler is working on doing something to a model, she may describe her work as "adding biology," and go on to explain the particular first-order mechanisms, in the form of variables and equations, that she is adding.

2.2.4 The Beauty of Box Models

Underneath the software implementation and the numerical approximation lies the conceptual articulation of dynamical processes, or the box model. Each box in a box model represents a distinct segment of the ocean that behaves differently: the segment for the surface, the segment for the deep; the segment for the tropics and the segment for the sub-tropics. The boxes are made by partitioning the original box horizontally or vertically, adding arrows with transitions, represented by equations. Nina, a junior graduate student[6], explained box models to me with palpable enthusiasm. She tells me that this is important for understanding the first-order mechanisms, which she cannot do with the large general circulation models (GCMs). Though initially skeptical, in practice she found that, indeed, those are too complicated, hard to understand. She tells me that GCM output studies need a section on validating with a box model, except for some of the huge IPCC ones. "If [my office-mate] was here, he would sing many praises to the box model."

---

[6]In the BioGeoChem-Model team, as introduced in Chapter 3.

Nina herself had only been exposed to the idea for a year, but she described them "amazing, like magic, didn't believe them at first." The box model is elevated relative to the programming aspect of modeling: "The coding is the easy part. Well, sorta easy, but you can do it in a day. Then you interpret it, and that's why you can write a paper on it." She first explains that, generally, you ought to use no more than 3 or 4 boxes and that gives you a lot of complexity and power already, but in the same breath she tells me about her "favorite box model paper" which "goes *all the way* up to 7 boxes!" The paper she shows is remarkable in particular in its high level of complexity, and the strength of its argument for why this complexity is necessary. Box models are ways to computationally and conceptually divide the ocean into a few elegant segments in order to focus on effects of particular dynamics.

2.2.5 Fieldwork and Interconnectedness

The primary subjects of study are conceptually interconnected. Additionally, the two modes of work (fieldwork-oriented and model-oriented) are procedurally interconnected. Models use observations to verify, observations are triangulated with other observations. In both cases, oceanographers proceed with caution and use as much of the available knowledge as possible to inform how the collection of environmental data is targeted, or how the model is parameterized. The key power of models is the capacity of the researcher to "take out" or "add" the first-order mechanisms; in this way, engaging with models offers the power to make causal claims. Though conceptually and procedurally interconnected, there are distinct values and practices associated with each.

Time is of crucial importance on board a ship not (only) because of a general preference for faster algorithms[7] but because observations in a cruise context happen on a schedule and are mindful of the water budget. A variety of instruments (floats that follow the ship,

[7]which is an intrinsically important feature of any algorithm in computer science, where the goal is not only to make some current task possible, but to make it so trivial that whole scores of other tasks become possible as well.

or a rosette lower form the ship) are used to collect readings and/or water samples from various depths at particular stations. In this paper, the authors mention damming; one of the teams[8] has an instrument that filters water as it passes through. Water samples can be filtered and/or kept in a fridge for analysis after the cruise. CTD (conductivity - which is directly related to salinity - temperature and depth) sensors are attached to the metal frame of the rosette, and the floats. At one point, as I was getting a tour of an oceanography building on campus, some technicians were outfitting a float in a break-out area. They explained that other sensors may be attached as well, relative to the needs of the research team in question.

On another occasion, during a brown-bag lunch with members an ocean-going group in the study, they explain to me that if they have data coming in from their instrument, it's up to whichever member on the team is on board to write impromptu scripts in any language of choice to aid decisions on tuning the instrument *in medias res*. A senior post-doc, George, tells me with excitement and pride of a recent cruise where the findings from the on-board analysis of their data provided valuable information for everyone on board, and was useful in choosing the route and timing of the cruise. It is not uncommon for decisions to be made in the course of a cruise that change the plan, though there is a detailed plan beforehand; to aid this, there is a Chief Scientist who balances pragmatic constraints and overlapping or conflicting scientific needs.

2.2.6 Time and Seasonality

Time plays a crucial role in both the content and conduct of research, whether it is observational or modeling. In the observational case, time constrains the sample analysis and filtering that can take place; especially if the research cruise has a schedule of visiting different stations, and if the researchers are interested in comparing their measurement across different points in the day-night cycle (the "diel" or "diurnal" cycle refers to movement of

[8]The CustomInstrument-Lab team, as introduced in Chapter 3.

certain organisms vertically, for example), in which case comparability between different days requires collection at roughly similar times during the day.

Many of the datasets and their associated displays extend over multiple months or multiple years. In this case, it is the cycling of seasons that introduces variation to be taken into account. Depending on the research question, it may be meaningful to compare years only by way of comparing the same season across years. Observational and model data alike can be plotted on these scales, sometimes together for comparison. The challenge of observational and model comparisons at finer-resolution time-scales, and of comparing between observations, is that if the measurement is interesting to look as it varies through the day, it begins to matter exactly how the different data sources are *normalized*. And when it comes to comparisons on coarser-grained but longer, even geological timescales, or forecasting into the future, observational data scarcity becomes a challenge.

In addition to ways in which time impacts the construction of oceanographic data as well as its representation, there are also familiar organizational seasons and rhythms: the addition of new group members, mentorship and teaching built into academic responsibilities alongside research, publication and grant-writing cycles. When it comes to the observation of code work, it is subordinate to all of these constraints. The maintenance of code in science can be seen as "extra" work (borrowing the term from Trainer et al. [62]), secondary to the other "scientific" (unlike code, despite its pervasive important [45]) duties. For example, the decision to use one programming language over another has, in the groups that have the most direct collaboration on code among members[9], been driven at least as much by technical considerations as reasoning about the hypothetical future students, or hypothetical future constraints of the senior researchers in managing their tasks.

[9]CustomInstrument-Lab and the RegionalNowcast-Model

2.2.7 Images and Communication

When the participants present talks to one another, for feedback or to share findings, there are photographs of the cruise and its crew, perhaps of the trawl net or the water-filtering instrument or rosette involved. Organisms are represented with beautifully-rendered images of diatoms. There are images of the area studied, which are colored in pixelated-looking shades of the rainbow, denoting the grid-size and providing an overview of the result, if they are produced by modeling.

The images of the area studied may also be satellite shots of a particular region, with squiggly line (or lines) marking the path of research cruises. Imagine slicing the 3D water-mass using the squiggly cruise-path line as a guide. A *transect* chart shows a a contour of the ocean floor along the x-axis and depth on the -axis (with 0 at the top). The ocean floor is not visible from the satellite, and individual cruises do not usually chart the shape of the bottom as part for the course.

2.2.8 Ways of Knowing

The *bathymetry* file that provides this data is one of the many instances of ubiquitous data re-use and triangulation of information from a variety of stakeholders and institutions in the field, as an interviewee[10] explains at one point: "Rivers have to come from a variety of sources, primarily U.S. Geological Survey. And then tides, that's easy. And then bathymetry comes from a hodgepodge of sources, often a NOAA database." As a computer scientist, my impulse is to accept that someone somewhere create sufficiently-reliable bathymetry understanding, and that if I were to use that, the provenance is best abstracted out. However, this kind of abstraction also removes the role of disciplinary differences in the ways of knowing, which are increasingly important when the knowledge in question is very difficult to build from limited data sources:

[10]From a set of interviews conducted with oceanographers, among others, by members of Charlotte P. Lee's group in spring of 2014.

> At the turn of the century [1900s], it was still possible to believe, and many geologists did, that Earth was shrinking as it lost its primordial heat; that, as it contracted, clocks of crust subsided to become ocean basins, while others were squeezed upward to form mountain ranges and whole continents; and that low points became high and high became low as the cooling continued and the erosive forces of wind and rain did their steady work on the land. On the other hand, it was also possible to believe, as the *Challenger*'s scientists had come to believe, that the present ocean basis were permanent features of Earth's surface. And finally, it was possible to believe, as a German meteorologist and visionary named Alfred Wegener believed, that continents drifted horizontally across the face of the Earth, ploughing through the ocean basins - which Wegener claimed were basically flat and featureless - as ships plough through water. [36, p.39]

Steinhardt and Jackson describe oceanography as a "radically unstandardized field ... that has struggled to converge on method and technique above the level of the individual site or PI research program" [60]. What became apparent in my observations is that the lack of standardization corresponds to a multi-disciplinary methodological pluralism. For example, the PI of one of the groups in the study[11] comments in a meeting, as she encourages a student to approach another researcher outside the group for advice on methods: "this is [the other researcher's] life's work, to make these databases [of sequences], and she's right here! ... I don't know how willing [the other researcher] is to work with our instrument, but if you have data in the right format, it shouldn't matter. She has a lot of experience and opinions about databases, and so do I," which she follows up by enumerating some of these differences of opinion. In each of the statements selected and highlighted here, the PI is (in order, corresponding to the statements/sentences): (1) seeking the assistance of an "owner" of a method, recognizing the other researcher as the relevant expert; (2) articulating the availability/accessibility in terms of physical or email proximity; (3) delineates the end of

[11]The Omics-Lab group, which is introduced formally in the following chapter.

an unshareable method (the instrument which the other researcher may not 'want' to work with) and the beginning of the shareable method (well-formatted data); and (4) neutrally recognizes the plurality of approaches.

Different ways of knowing inform the content of the claims an narratives made by different kinds of oceanographers. In the anecdote with which I opened this chapter, excerpted from *Mapping the Deep*, Kunzig attaches the theories to people, and points out their backgrounds (geologist; scientists previously introduced with the Challenger crew; meteorologist). Talking with Mallory, she mentions that she came to two researchers at UW with a question, and it "left me sort of reeling: interesting conversations but unclear how to proceed. They gave essentially completely opposite answers," depending on the scale at which they commented. Previously, she had pointed out that the scale of physical modeling (often, using 1 by 1 degree grids) is far too coarse to effectively include biology, because biology changes over smaller geographic ranges. Her questions persisted: how far to generalize? How to make sense of patterns in nature?

Continuing his story about bathymetry, Kunzig notes that "it was possible to believe many things about the ocean in the early years of the century, because so few things were known. In 1904 ... the newly formed [bureau] had prepared the first standardized bathymetric chart of the world ocean, collecting all the available soundings. There were 18,400 of them. That was not enough to understand the ocean floor, and in fact geologists did not begin to understand the overall architecture of the ocean floor - or the planet as a whole - until half a century later. Until then, they really did not know what they were talking about." However, knowing that "you do not know what you are talking about" requires the wisdom of hindsight. In the moment, all a scientist has at their disposal is their varied skills, their research intuition and taste, and their vigilant skepticism and striving for control and transparency whenever possible.

2.3 Summary

Code and computer work can challenge to what it means to "do science." Jackson and Barbrow argue, based on an ethnographic study of cyberinfrastructure project in oceanography that "some of the most profound and important questions around computational change in the sciences may in fact be vocational in nature, producing basic changes in the sense of what it means to 'be' an ecologist, and to 'do' ecology" [31]. Scripting is increasingly common in ecology [44]. Sennett elaborates the notion of vocation, of "craftsmanship," as being "dedicated to good work for its own sake" where labor elevated above being the means to another end [57]. Though it is temping, as a computer scientist, to take the aesthetics and craft of code work for granted and to see software tools and skills as a means to greater efficiency of discovery in science, experimental software (e.g., the cyberinfrastructure project, or a tool arising from a collaboration with scientists, such as Grochow et al's hybrid client-and-cloud visual analytics program [24]) demands more than just patience and attention from the target users, but engagement with code work itself, which can imply a major qualitative shift in work.

On the other hand, it may not be nearly as transformative in contexts where programming has been an integral part of scientific practice. These existing code work practices, though, can sometimes be perplexing from a computer scientist perspective, which sees ways that this code work is unnecessarily painful (e.g., debugging and testing is difficult because abstraction is minimal, and design is not a distinct part of the process), and would (implicitly or explicitly) recommend adoption of particular best practices [27, 58, 70, 38]. Heaton and Carver, in a review of literature that makes claims about software practices in science, distinguish "kleenex" software (ad-hoc, single-use), from "library/resource" software in the sciences, specifically in computational modeling [27]. Trainer et al., in studying motivations of scientists to undertake "extra work," consider larger projects of the "library" variety [62], as do Paine et al. in their study of code co-production with data in a large radio astronomy collaboration. Sometimes "kleenex" software is equated with ad-hoc, non-reproducible code

practice that is implicitly inadvisable and, in a perfect world, would be replaced by better, more holistic analytic tools (e.g., [29]).

Chapter 3

OBSERVING HOW OCEANOGRAPHERS CODE

The hypotheses, theories, and frameworks in this dissertation are based on observation of individual scientists grappling with code. Code work constitutes only a part - sometimes a very small part - of the overall range of the daily work done by the scientists who participated in the study as informants. "Observing" code is just about as quiet as it sounds: I am in a corner taking notes. A scientist (Erin) is at her workstation. She runs a script in terminal and sits back, watching as image after image comes on-screen. The images are most often either line plots with multiple lines for making comparisons, or rainbow-colored maps of the geographical region of interest. Each new rainbow-map with different bright blue (or red) blobs in different places, indicating different temperature gradients produced by running the model with different parameters, or for longer time spans. The scientist sighs and looks at the screen with a furrowed brow. "What are you doing now?" I ask. "Oh," she answers. "Trying to convince myself that it works."

I ask her follow up questions either in the moment or during a coffee or lunch break, having made a note of the event but not wanting to pull her out of her creative zone. I ask her what would make the output "convincing," which "it" is in doubt, and what is meant by "working." Her answers identify some code components as *persistent*: reliable and robust over time. Other components are *ephemeral*: with the expectation of imminent degradation, such as if the idiosyncratic data format of a third-party site for which a script is written suddenly changes. Additionally, code can be *brittle*: uncertain in an unclear way; inspiring the expectation of breaking, but without a clear enough idea of how it might break that a preventative measure might be taken. The terms *persistent*, *ephemeral*, *brittle* are not often stated by the participant, but they are reduced to these labels as I iterate through my notes.

Some code components are implemented the way they are for "historical" reasons: a prior colleague had preferred a particular working environment, and this had specific, tenacious, possibly vestigial ramifications. If the scientist had more time, she would change this or that component, but she does not, so she thinks aloud about what circumstances would create sufficient pressure to make the change. Yet another component used to work, but might not now, because the group has upgraded to an updated version of a core software and it is not yet clear how that has affected this component. In this way, the properties of the code are contextualized in an interpersonal and historical narrative. The goal of the analysis is to identify patterns in these code narratives in different groups, and how they affect deliberate change.

In this chapter, I introduce the methods (observation and interview), the site (oceanography), and the participant population (the particular oceanography teams studied). By the end, the reader will have both a sense of the scientific and social context of code work, as well as a clear idea of concrete study procedures.

3.1 Methods

All data were collected using a combination of interviews and observations. The observations focused on either (1) events and meetings, such as regular lab or group meetings and special programming skills workshops, or (2) the mundane daily activities of particular researchers, whom I was "shadowing" that day. Though up to a hundred scientists had cumulatively been observed over the course of the various events of interest, the core of the study is comprised of 21 oceanographers across the groups who were actively adopting/adapting new programming skills over the course of at least a year. An *additional* 25 oceanographers were in the same groups and interacted frequently with the core 21 in projects I included in the study. Seven out of the total 46 oceanographers were also interviewed about, and initially recruited through, the Software Carpentry (SWC) workshops, in addition to 13 scientists who are not oceanographers. The study participants are summarized in Figure 3.1. The figure illustrated the dual inclusion criterion and overlapping recruited populations.

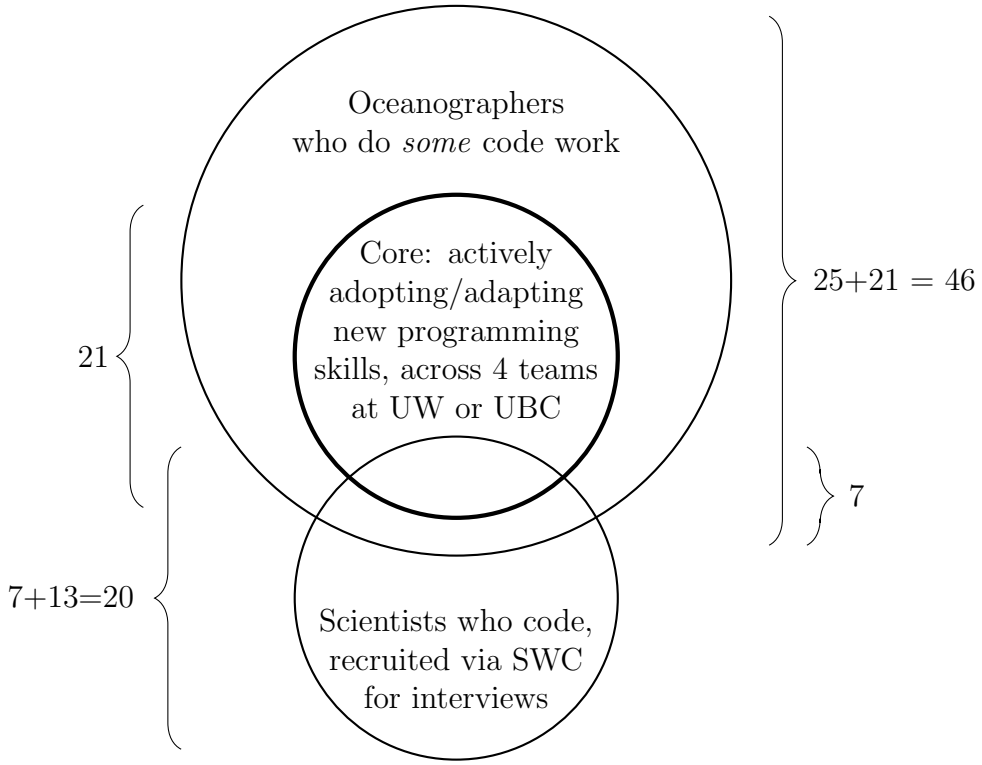

Figure 3.1: **Study participation summary, grouped by inclusion criterion.** This figure illustrates the overlap between participant pools of scientists interviewed after attending SWC-like events, and oceanographers recruited via SWC and Data Science Environment at UW.

Data collection was conducted in three stages over the course of 18 months, from April 2014 to December 2015. The study was designed to get a breadth of examples of software adoption and adaptation, as well as a breadth of individual and interpersonal narratives of code-related decisions. First, I recruited and interviewed SWC participants. Second, I recruited the four teams that comprised the core population and did initial fieldwork by attending regular meetings and events. Third, once I had a better idea of the various projects in each team and a sense of the group dynamics, I conducted the bulk of my observations by "shadowing" each of the core participants anywhere between several hours and multiple days. Because my time was split between the various teams, I conducted "check-ins," or

very informal interviews initiated over email, or casual half-hour chats in a hallway, over brown-bag lunch, or over coffee.

Only the first set of the SWC interviews, as well as the reflections of the team which did the least code work, were audio-recorded. The audio recordings were transcribed and added to the dataset. All other interviews and observations were in the format of handwritten field-notes which I then typed up. Occasionally, these field-notes included excerpts from various emails, such as a "meeting summary" sent out of a joint meeting, or an interestingly-worded event reminder that helped to understand the context and significance of that event. The data is around 300K words[1] in total, though the density of words varied as my initially verbose style of note-taking became more condensed and focused over time.

The original recruitment in the first stage was focused on Software Carpentry participant. Twenty (20) participants were each interviewed between 1 and 3 times over the course of several months, depending on the extent to which code work and new programming skills were a part of their daily activities following the event. Of those 20, 7 were oceanographers from 2 different groups that were then included in the study. During the second stage, I reached out to both groups and included them in the case study of oceanography. The remaining 2 of 4 total oceanography groups were included based on their engagement with the Moore-Sloan Data Science initiative on the UW campus. In this way, the **inclusion criterion** required all study participants to have actively engaged with a particular zeitgeist of increasing/improving coding skills in scientific work. All had spoken of "best practices," though mostly in a context of *striving* rather than *currently doing*. By "engagement," I mean not unanimous embrace, but a collective sense of "this is the way things are going," as one SWC interviewee commented in explaining her decision to spend time learning Python despite not having a clear, immediate use for it. In other words, not all participants were un-

[1]All of which are stored in private git repository in plain text with a metadata header and verbose filename. No qualitative analysis software was used. I had written a few simple python scripts to process the data, but ultimately relied mostly on grep and memo-writing for all analysis tasks. Some images were stored in a folder in the repository and there were not so many of them, and they were not so big, as to warrant any specialized process.

equivocally enthusiastic about the increasing amount of programming competence expected or required for professional success, as many had reservations.

3.1.1 Group Events

Workshops, "skillshares," "sprints," "joint meetings," and other such events are special in that they allow their interdisciplinary scientific participants to explore and enact their collective imagination of what is possible in the field. The idea of the collective imagination will be developed further Chapter 4. Figure 3.2 shows the different events. Here, "work" refers to "code work" but overlaps with the practice of science more broadly. Events included span both coding "sprints" and guest lectures because code is not readily separable from its scientific context. In theory, idea is introduced in Section 2.1, particularly using [32, 45]. In practice, any context where scientists explained findings to each other that involved any of the many possible computational interventions was an excellent opportunity to study narration of code work. Table 3.1 shows the amount of observations and interviews in the data set as a whole, including both oceanographer and SWC-specific populations.

By distinguishing *closeness* of work, I try to highlight the range of whether multiple people are closely engaged with the same piece of code at the same time. For example, being around people to be encouraged, share momentum, or be exposed to ongoing scientific work for general awareness is on the "low closeness" end of this axis. In this case, the code written can be an extension of a single individual's thought process, and follow whatever standards that person holds their code to. The "medium" case includes, for example, asking for help or feedback when needed, where the code written by an individual *does* become part of communication, feedback, or negotiation (reaching out for "high-level feedback" involving no code would be "low," not "medium" on the "closeness" axis). The "high" end of this qualitative spectrum involves working closely with at least one other person who also has a stake in that particular work, which requires all those involved to negotiate (or have, in the past, negotiated) a consensus across the differences in their working practices.

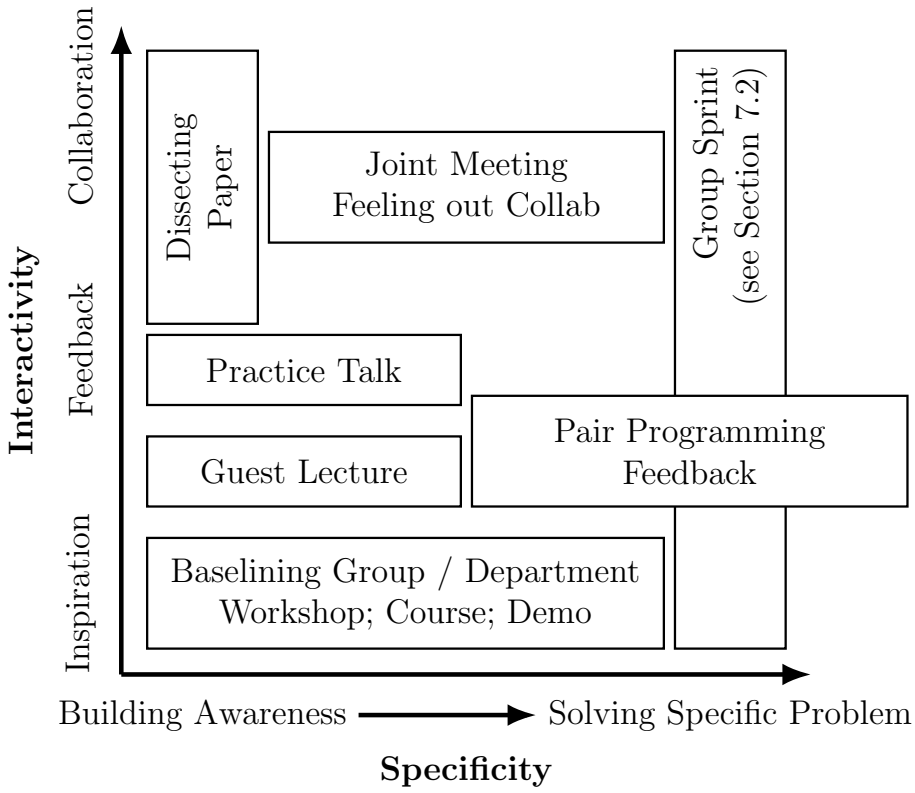

Figure 3.2: **Summary of the different *group* events included in they study, distinguished by interactivity and specificity of work.** *Interactivity* refers to the extent to which multiple people are actively working together on the same piece of code. *Specificity* refers to how explicit, articulated, and/or pre-determined the aim of the event is. Excluded from this chart is "shadowing" which focuses on the individual, though may involve also participating in some of these events.

Type	BGC-M	CI-L	O-L	RN-M	SWC/etc
Joint Meeting	1.5 (3)	8.5 (8)	13.37 (9)	1 (1)	N/A
Demo (+)	1.5 (1)	1.5 (2)	2 (2)	2.5 (2)	N/A
Feedback (+)	2.5 (2)	3 (4)	2.5 (2)	6 (5)	N/A
Talk (inc. practice)	6.12 (4)	2.25 (2)	2 (2)	5 (3)	N/A
Talk (practice only)	4.65 (3)	1.5 (1)	1 (1)	1 (1)	N/A
Shadow	57.08 (8)	28.25 (4)	4 (1)	36.75 (2)	N/A
Casual (+)	4.95 (7)	4.5 (5)	0.5 (2)	0.5 (1)	N/A
Interview	0	N=3	N=5	N=3	N=17
Workshop	3.5 (2)	3.75 (2)	0	19.5 (4)	9.25 (3)

Table 3.1: **Qualitative data set summary.** A breakdown of data collection across the four groups and over all the different event types, which totalled just over 300 hours, and excludes the interviews and observations conducted by my colleagues that were pertinent to the study and included in the analysis. All amounts are listed in the format "hours (instances)" - so, for example, "2.5 (2)" means that a total of 2.5 hours, over 2 different occasions, was observed of that event type (row) for the group (column). The interviews are listed by count rather than time measurement. They lasted between 30 minutes and 3 hours, and the cumulative total time is not meaningful to compare across the columns. "+" means that this was a pervasive dynamic throughout many casual interactions and therefore the given number is a conservative amount. Across the top, I included short-hand labels for the 4 oceanography groups in the study, introduced in Section 3.3 as BioGeoChem-Model, CustomInstrument-Lab, Omics-Lab, and RegionalNowcast-Model groups. The participants interviewed about their experiences with SWC-like events are also included. Note that the "Interviews" do not include ongoing casual check-ins. Of the 20 interviews for SWC, 7 are listed under O-L and RN-M, reflecting the overlap in participant populations shown in Figure 3.1. The interviews in group CI-L refer to more structured interviews with team members whose role had changed, on the subject of that role change.

The extent to which this work is *targeted* refers to how much the work is aimed at a particular goal or project. For example, a group sprint by one of the teams[2] spanned a variety of possible tasks and had room for suggesting other additional undertaking, but all the projects were in service of a stakeholder-facing web visualization dashboard that existed in addition to individual researchers' scientific agendas[3]. I distinguish this from the "Joint Meeting," or "Feeling out [a Possible] Collaboration," which are meetings 1-on-1 between PIs or more senior researchers, or between multiple members of different groups, with the aim of considering possible overlaps and mutual motivations. This is less targeted than working on a joint artifact as a group, but more targeted than practice talks or dissecting a particularly interesting paper, which can exist without directly contributing to an active line of research. For example, seeing a guest lecture may not have any immediate application to a specific ongoing task, or be only a open issue; however, it is an essential part of the overall practice of doing science. Demos and workshops for "tangential" tools of skills may be an immediate distraction from a task at hand, while helping raise awareness.

The separation of events into these categories helps us to have a feel for the mood: a paper discussion is fun, exciting, inspires curiosity, and not generally undertaken while a group is rushing to meet a deadline. A "pair programming[4]" session, or a meeting between 2 or more people which involves working closely on a single code-base, has the potential for much more urgency or frustration. This is the more "problem-solving" kind of "event." Events like talks and paper discussion are included - they are not centered on code, but given the ubiquity of programmatic means to accomplish a great variety of tasks, issues of trusting or re-implementing code come up in these events and are included.

---

[2]The RegionalNowcast-Model team, which is introduced in Section 3.3

[3]This was a major data stewardship deliverable of the group, but was not itself the aim or domain of any particular researcher

[4]not generally called this even if it looks like it; grouped with 'expertise exchange' because the reason for these is often explicitly to pull in someone who has additional specialized knowledge. I return to "pair programming" as a concept and a term in Chapter 5.

3.1.2 Observing Code Work

As shown in Table 3.1, I conducted 28 semi-structured interviews during this study; the protocols for these is included in Appendix A. As is typical in semi-structured interviews, most of the interesting questions come as follow-ups. Aside from 28 "formal" interviews - scheduled, with an audio-recording device, in a purposely private office - there were countless more casual interactions where I could ask the salient questions. Additionally, toward the end of the study, I asked speculative questions. For example, during a lunch break, I might ask the participant to critique an idea for interactive visualization for debugging. The goal of these probes was to inspire additional reflection, in a concrete way, about a larger issue, like trusting automation. Often I would summarize what I saw that morning in observation, or a small portion of the working insights so far, as a kind of sanity check. Most formal interviews were audio recorded and transcribed, though many insights emerged from the informal lunchtime "interviews," which I transcribed to the best of my ability in observation fieldnotes.

In addition to formal and informal interviews, I conducted 126 hours of observation of code work, the breakdown for which is shown in Table 3.1 (304 is the total hour count, and 126 is the total for "Shadowing"), which excludes interviews, check-ins, and discussion of code in meetings or events. As noted in the beginning of this chapter, "shadowing" consisted mostly of quietly co-working in the same space at the scientist being "shadowed" and asking periodically what they are stuck on, what solutions they are trying, how they know if it is working or when do they decide to quit and try a different approach, and so on.

The choice of the word "code," rather than "software," "programming," or "scripting" is intentional. "Code" refers to the *programmatic instructions* written in the course of software production (and sometimes software use), programming, and scripting, though these are distinct coding activities, which I refer to as different kinds of *code work*, as I categorize and illustrate in Table 3.2. This table includes both a typology of code work from existing work [27], and another one that reflects the typology of the code work I studied. My typology

includes “goliath” as a term used by a study informant and referring to especially large modular codebases developed by scientists for scientists.

Kleenex	Scripting	processing data, coping with formatting
		Exploration, iterative visualization, analysis
	Modeling	Using model
Library		Adding to model
	Integrating new package	Documenting code
		Verifying whole pipeline
	Goliath	Making something into a module
		Setting up / using pipeline
Data Stewardship	Design and implementation of endpoint for data access	

Table 3.2: **Categorization of code work observed.** The categorization of “kleenex” and “library” maps onto the distinction embraced by Heaton and Carver [27]; data stewardship can be exemplified in the Ocean Informatics work [4], and “goliath” refers to major programs that are programming environments that enable constituent libraries and packages, with the phrase borrowed from a participant story included in Section 4.3. The arrangement of the categories and distinctions made reflect the focus of this study on the individual experience of doing code work - writing from scratch or modifying? Open-ended approach considering multiple possibilities, or skeptical and interrogating a recent implementation of one such possibility?

3.1.3 Biases and Researcher Stance

One of the 4 groups studied was housed at the University of British Columbia department of Earth, Ocean, and Atmospheric Sciences. The other 3 were housed at the University of Washington School of Oceanography. In the course of collecting data, I travelled a dozen times to Vancouver, BC for observations and interviews spanning a year with a particular

focus. As Miles, Huberman, and Saldañ a point out, one of the sources of bias in a qualitative study is the effect of the research site on the researcher [43], such as when the researcher becomes too close to the meanings and values of the informant. This kind of "going native" is arguably especially tempting in the study of science [37], as both the researcher and the informant share certain aspects of their creative endeavors. Although it was not intentional, the physical, temporal, and emotional distance between Vancouver and Seattle allowed me to gain unanticipated but welcome perspective on all the study sites. Indeed, Miles, Huberman, and Saldañ a recommend "spending time away from the site; spread[ing] out ... site visits" as a way to avoid this form of bias [43]. Physically changing location between Seattle and Vancouver, including the multi-hour bus and bike "commutes," inspired almost all of the most interesting insights reported in this document.

I worked to mitigate the effects I might have on the site as a researcher, because they seemed to loom especially over this study. My programming background, and my affiliation with CSE, allowed me to observe the kinds of *code work* I described in the previous chapter at a fine level of detail. Occasionally, the participants asked me for an evaluation or assessment of their code or code practice, with different degrees of seriousness. I deliberately did not put myself into a "software advisor" role (see Subsection 3.2.3), except by accident in the story of a group trying out Tableau (see Section 6.4). Additionally, I believe that more often statements like "I should really [be doing this thing] but I'm not" may have been entirely for my sake, an acknowledgement that their practice deviates from some protocol but not necessarily with any guilt, shame, or anxiety about it. When I encountered these kinds of reactions, I asked further questions, in line with the advice to "asking [the informant] to be attentive to your influence on the site and its inhabitants" to avoid the bias of researcher effect on the site [43]. For example, I asked why they think that they "should" or "should not" do something, and the reasons for their choice to diverge from the perceived expectation or standard. I would further reassure the informant that my purpose is to understand their programming practices, rather than to judge or evaluate.

In additional to the potential biases in interaction with the informants or with data collection, there are potential biases in analytic approach, which I addressed by seeking feedback from non-computer scientists throughout the data collection and iterative analysis processes. My computer science training offers me a vocabulary for describing things and events; it also offers an implicit and abstract set of ideas about what is or is not "elegant" or even "good." My aim as a qualitative researcher was to use the knowledge I had to be able to understand what people were working on, but to recognize and set aside my learned judgments as much as possible.

Below, I describe in more detail one such example, involving the terms "bug" or "debugging." This is possibly one of the least-used words by participants. Indeed, I have a strong suspicion that on the few occasions the word was used, it was because I used it in a question without catching myself. Rather than calling things "bugs," most participants said that their code "is not working" or "being weird." Rather than "debugging code," they would re-examine the code as well as look over the data (if any) in plain-text or in an overview visualization, or re-do the math (if any) in pencil-and-paper or whiteboard.[5] In an overwhelming majority of cases, the bug had to do with assumptions in parsing and formatting of dates; unit conversion; or a sneaky divide-by-zero error arising from the current coder making assumptions different from the previous coder on the range of some variable(s).

One way to interpret the apparent lack of calling debugging by its name is to see it as symptom of a problem, and to suggest a solution. These non-expert programmers, without formal training, are not fully appreciating the extent to which the implementation of an idea (e.g., a system of differential equations) is distinct from that idea: checking the math gets you only part of the way to the bug (this is one of the points made in [27]). The resulting recommendation could involve suggesting that scientists, at workshops like SWC, are encouraged to speak more about code as a separate entity, with its distinct opportunities

[5]Almost every time my follow-up questions about a bug were not easily answered with a few phrases, the participant grabbed a nearby pencil and paper, or marker at the whiteboard, and sketched something explaining the inter-operating components - invariably not *only* in the code but outside of it, as well.

for brittleness, e.g., by formally acknowledging "debugging." Alternatively, we might recommend doing pair programming with a more experienced software engineer who can guide the inquiry to possible holes in the code.

Another option in this situation would be to acknowledge that it is hardly surprising that people focus on what they are most familiar and comfortable with in problem-solving. Borgman describes a similar effect between the "science team" and the "technology team" [12]; and Patel et al have written about the reverse effect: the fact that programmers focus more on the code than on the data makes it more difficult for them to train good machine learning models [47].

The difference between the first interpretation and the second is the difference between critical, evaluative assessment and ethnographic study. Another way to look at this difference is to contrast the terms *etic* and *emic*. The first refers to concepts and terminology applied by outsiders to a group of people and their activities. The second refers to the concepts and terms as they are understood by the group of people themselves. Being surprised by a lack of "debugging" is an etic judgment. Moving away from "debugging" and focusing on the mechanics of problem-solving in the terms of the scientists themselves aims to understand the emic values and meanings.

Software engineering literature has used literature reviews [27, 58] and well as surveys and interviews [26, 34]. Work by Adolph, Hall, and Kruchten translates grounded theory methods for software engineering practitioners and makes a case for it as a valid means for building knowledge [1]. Advocacy for deeply qualitative methods in computer science research and system design dates back for decades (e.g., [21, 2]). Furthermore, working closely with particular teams to design and validate interventions is a recognized practice in CSCW as well as in software engineering (e.g., [74, 46] respectively), though for historical and community constituency reasons, formal qualitative methods are far more common in the former. The use of qualitative methods in this dissertation is therefore not groundbreaking in computer science. The more unique turn that this dissertation takes is the simultaneous recognition of the idealized software engineering concepts (etic) at the same time as aiming

to contextualize and describe the participant's own meanings and values on the relevant subjects (emic).

3.2 Oceanography Participant Population

In this dissertation, the focus is on researchers who study dynamics, processes, and small particles and micro-organisms in the ocean. For example, even the graduate student who was working on a modeling project regarding the overfishing problem is embedded in an epistemological environment that concerns itself with changes of concentration over time, with movement assisted by eddies and currents.

I studied four teams, which I refer to as the BioGeoChem-Model, CustomInstrument-Lab, Omics-Lab, and RegionalNowcast-Model groups. This delineation does not preclude a diversity of backgrounds of group/lab members as well as a diversity of interests. Figures 3.3 and 3.4 provides an overview of all the groups by way of justifying their **pseudonyms** and roughly introducing the **participants** involved in context of the relevant **inclusion criterion**. This section focuses on contextualizing these components of individual groups in the various technological resources and ongoing challenges, some of examples of which are offered in this section.

3.2.1 Informant Teams and Inclusion Criteria

The two Model groups and the two Lab groups are on the two sides of the modeling and observation distinction that is described in greater detail in Section 3.2. That said, both groups have members who have either moved from one "side" to the other or are combining/bridging to some extent. Their remarks were used in Section 3.2 to characterize the distinction, but also underline that it is not so rigid and unyielding as my group nomenclature might imply. The BioGeoChem-Model and CustomInstrument-Lab groups were included because each had at least one lab member engaged in a data science initiative on their campus. The other two groups were included because I encountered at least one of the group members at a Software Carpentry event.

	BGC-M.	CI-L	O-L	RN-M
Inclusion	Data Science Environment		Software Carpentry Workshop	
# Participants	11	15	9	11
# Core Part.	7	4	4	6
	Interest in Programming and Core Challenges			
Analysis	Transformations and Comparisons	Faster, More Data	More Automation	More Features
Visualization	Interactivity	Real-time and archive site	-Omics big data	Daily forecast site
Big Data	Access	Storage	Handling	Access

Table 3.3: **Summary of interest in code work among oceanography participant population, categorized by group.** In addition to inclusion criterion and participant count (total and in the core group), this table provides a summary of the **core challenges** that the teams have in mind as they explore various interests in programming.

	BGC-M	CI-L	O-L	RN-M
Closeness	low	pairwise	low	high
In-house S.A.	Cluster, Modeling	Web, Visualization, Scripting	N/A	Analysis, Web, Vis.
External S.A.	Vis.	Analysis, Data	Analysis, Hardware	Cluster
Other tech	Cluster (modeling)	Custom hardware (ship)	Lab hardware	Cluster (modeling)

Table 3.4: **Summary of interpersonal code work resources available to the oceanography participant population, categorized by group.** This table summarizes the code work context, in terms of constraints and resources. First, **closeness** refers to closeness of code work in the same sense as included in the Figure 3.2. This is a sweeping summary of overall dynamics, and furthermore only includes *code* work (e.g., the Omics-Lab would have "high" closeness of work if I included lab work, as well). Second, **S.A.** stands for "software advisor," and this role is described more in Subsection 3.3.3. The "In-house S.A." row includes a brief description of the kinds of work that S.A. does, such as handling a computing cluster, visualization work, or data management. The "External S.A." refers to a collaborator or staff that is somehow less accessible than the in-house S.A., or in more specialized or dire situations. The last row includes an overview of other technology that is not covered by my account of code work but which is a major part of that group's activities.

The "working group" or "team" was either the complete "lab" of the particular PI, including graduate students, post-doctoral researchers, and sometimes research staff; or the group surrounding a particular shared technological artifact. The CustomInstrument-Lab and RegionalNowcast-Model groups fall into the latter case. The CustomInstrument-Lab works on developing a particular, new hardware instrument for data collection during cruises. The RegionalNowcast-Model group develops, among other things, a website for a variety of local and international stakeholders that shows their short-term forecast results. These include storm surge warnings, with consideration for stakeholders (e.g., displaying things in knots rather than m/s); as well as extensive documentation, as described in the story below, *The Upgrade*. Each group has ways of knowing and areas of interest both in scientific and code work, illustrating which is the goal of the stories below.

3.2.2 Staying Close to the Reality of Data: The Pile of Salt

The two Model groups spend most of their time in front of their computers, and the two Lab groups have laboratory space and equipment where they also spend a great deal of time. The latter also go on research cruises where they bring their laboratory equipment to carry out specialized analyses; as well as bringing water samples back to shared fridges. Neat charts summarizing the process of science, such as in Philip Guo's "data science workflow" [25], or my own Figure 2.1, have a way of abstracting "data collection" from the realities of these data collection objects that, in actual discussions among study informant, are grounded in the decidedly, strikingly banal. For example, one of the Lab group members remarked that on the cruise, their fridge space will be limited because one of the collaborators refuses to share their fridge, because in the past there had been an issue with non-researchers on the ship also storing in it the fish that they had caught to eat.

Another post-doc, from the CustomInstrument-Lab team, showed me during lunch photos on his smartphone of barnacles growing on a data-collection instrument on a ship. This example was used to demonstrate to me the idiosyncracy of real-time data availability on ships, and the particularities of the technicians on board, or "marine techs". This particular

marine tech was persuaded to open the lock-box (revealing the damning barnacles, which had been ruining all the data for apparently a long time) by the scientist's observation that "we were getting Puget Sound [very high] levels of chlorophyll" in an area where nothing was living at the surface, so there should have been far less chlorophyll. A great deal of the cruise-related conversations among members of the Omics-Lab had to do with shipping samples, packaging them for mail, or taking them on the plane through airport security. In addition to handling samples post-cruise, they discussed the physical ability of the individual lab member going on the cruise to carry all the equipment on board, and to run the samples at each of the intervals of 4 hours. Discussions beyond cruises had a similar degree of grounding. Talking at some point about another group's protocol, a student remarks: "'12g of salt?' Have you *seen* 12g of salt? That is a *pile*!" Although abstraction of concepts is necessary for the study of ocean as a complex system, group discussions are deliberately grounded in the concrete.

3.2.3 The "Software Advisor" Role

The two Lab groups are not situated in the same building, but they have some collaborative connections. Members of the groups go on cruises together, along with members of other groups not included in the study. One of the cruises organized during the course of my observation had a whopping 16 PIs, and many of the discussions about logistics revealed a lack of effective communication that was the subject of both direct and indirect expressions of frustration. However, cruises are the source of much "precious" data, and one fruitful outcome is the subject of "The Beautiful Chart" story in Section 3. The communication difficulties were exacerbated by not being physically housed in the same building, but across campus. The SA role is earned and recognized by being approached for guidance and feedback on technical issues.

During one of the shadowing sessions with a member of CustomInstrument-Lab, Andrew, we went out to a picnic table across a bridge from the building to eat our sandwiches. Another researcher, Allen, joined us. He was not part of the team I studied, but he was

mentioned by the other participants and appeared occasionally in events I observed. Andrew often plays a software advisor role for the CustomInstrument-Lab. I use the label of this role liberally, referring in alternation to: (1) the act of "advising" a decision based on a series of discussions with those who will be most affected; (2) close work on a single technological artifact with the person that is it "for" - the person who "owns" the scientific aspect embedded in the software artifact, and who is the primary intended user - and, lastly (3) creating something explicitly for the scientific target-user with limited or no collaboration (outside of user feedback) from that target-user. A software advisor's legitimacy lies in other members of the group approaching that person for specific sorts of advice. I use the term "role," rather than "communities of practice," to describe these actors because it was not their values and backgrounds that made them software advisors, but rather having been collaboratively appointed to hold a certain decision-making and gatekeeping position.

Andrew was all-of-the-above at different times for the members of the CustomInstrument-Lab, though he, too, self-reportedly lacks "real" computer science training. Allen did not work with the CustomInstrument-Lab directly, but provides Andrew advice, of the (1)-type Software Advisor variety. He was in the process of pursuing an oceanography degree and had a formal computer science background. He also spent a great deal of his time supporting and maintaining a software project used by Andrew and other scientists. When Andrew, Allen, and I sat at the picnic table, they told me how you can tell that the field is changing just by being in their building. Their floor was designed less than a decade ago, but the computer rooms and offices face the less-picturesque nearby road, while the lab faces the beautiful lake, because their PI wanted to incentivize people to be in the lab more. Other time, the reality of it became that people spent more and more time at the computer stations. Besides, those running photosensitive experiments in the lab were not interested in the beautiful sunshine and kept the blinds down. They also told me that their presence is an artifact of the resources this PI has for experimentation: she is able to allow them to do their risky software projects that might not pan out, and that is not a privilege every lab has.

3.2.4 Physical Spaces

The two Model groups are not in the same building and they had no collaborative connection I witnessed in the course of fieldwork. They were using a different model code, and focused on different geographical regions. The members of each group spend most of their research time interacting with software, with other members of thir own group, and possible collaborators cultivated by individuals rather than by the group. Despite not spending much time with water samples or the ocean itself, both groups were deeply personally connected to their surrounding environments[6]. Both had at least one member who had never "been to sea," but who was very interested in taking any opportunity to do this, if it was relevant to the work. When one of the members of the BioGeoChem-Model group went on an education- and outreach- oriented Antarctica cruise, it was a major and exciting adventure, with enthusiastic stories told around brown-bagged lunch afterwards.

On one occasion, during a presentation of modeling research results, members ofthe BioGeoChem-model team got derailed. Jack, a post-doc, was giving a practice talk where the main overview slide[7] has a spinning animated 3D transect[8] and bathymetry chart of world ocean. The PI remarked, thoughtfully, looking at the chart: "hmmm... we lose perspective on how shallow the ocean is..." The presenter jokingly responded, "sometimes I lose perspective on what the ocean even *is*!" and Caleb, always enthusiastic to amplify a joke, mentioned with feigned bravado that he swims in it every day. It is late Friday afternoon, and Carl, a post-doc, keeps the joke going, exclaiming in mock-astonishment: 'why?! it is

---

[6]This sharply contrasts the scientists that Traweek describes in *Beamtimes and Lifetimes* [63], whose highly theoretical high-energy physics research was physically situated in on a campus that was at odds with its environment, and was undertaken by people who took pride in transcending their physical location. The physicists in my study talked frankly about work-life balance, about hiking, about their love of the outdoors, and about their interests in bicycles and boats. Parts of this reflects the outdoors ethos of the Pacific Northwest at large.

[7]The previous chapter introduces some common elements of visual storytelling in this setting, including an "overview slide" which is distinctive in including the most zoomed-out variant, with the most dimensions included.

[8]A plot with the y axis corresponding to depth (0 at top) and x axis to distance along a path, such as a cruise path, in the 2D ocean dimensions.

full of various particles!" Lastly, the presenter, Jack, wrapped up the the joke before moving on: "yeah, just a vat of metals and fecal matter." The overall rapid-fire, casual exchange got a lot of laughs and engagement. This story underlines the extent to which the participants see the focus of the research as a constrained abstraction of an exceedingly complex system, and the extent to which *failing* to maintain a sense of perspective causes misunderstanding.

Section Summary

As a whole, this section illustrates the extent to which discussions observed in this research *continually strive to ground the abstract in a practical reality.* In this setting, measurements ("12 grams of salt"; concentration of chlorophyll, metals, or fecal matter) are not agnostic of the actual or potential human experience of those numbers ("a pile!"; photograph of barnacles on a lock-box ruining all the data; a joke about swimming in the murky ocean).

3.3 Programming Interests and Resources

In this section I describe what the different teams work on. I use key stories to introduce the teams' social dynamics, research interests, and programming resources, as summarized in Figure 3.4. A shown in the figure, the four different teams have different resources when it comes to access to someone willing and able to play a software-advisor role. I distinguish between the in-house and the external individuals. As in the previous chapter, the descriptions are contextualized in stories that aim to portray the overall group dynamic in which code work and collaboration are embedded. the introductions below are selected to highlight representative interactions, setting the stage for subsequent stories.

3.3.1 The Best Meeting Ever

When I approached the BioGeoChem-Model team about participation in the study, asking whether they had regular research meetings I could attend for observation, I was met with a response that did not place much emphasis on the regular meeting. The weekly meetings

that started up and which I began attending were very casual. In addition to these, there was an almost-daily lunch that provided a common point of contact and where all the researchers, the PI, and occasionally other faculty, research staff, or students, would join. At these, conversation shifted smoothly from casually personal (about bicycling and weather) to the technical (weather predictions and model grids, or visualization color scales and color-blindness) if an interesting topic was found. This group is interested in programming chiefly for visualizations, as described in more detail in a "joint meeting" discussed in Section 7.2. The group members work independently, collaborating chiefly through feedback on visual artifacts and through comparing results to results of other groups.

A few weeks into observation, a meeting that was casually scheduled for Friday afternoon had gone on for an additional hour. The only structure it had lay in the joint paper that was under discussion. This group typically had meetings that either had no structure and followed a show-and-tell model, or were structured around a single major piece of scientific content, like a paper or a practice talk. At the end, this meeting was literally referred to as "the best meeting ever". Over the ensuing months, members of the group continued to work on this "side project" on and off, collectively and with a lot of enthusiasm. The project, anchored on a particularly intriguing lab study finding, captured the collective imaginations of the diverse group members. Months later, the simmering "side project" was brought up in a teleconference call of one group member to a collaborator of his. When I observed individuals, working on this project was "fun," and something that they would do as a group sitting around the common space of the building in the tired and sunny afternoons. In other observational contexts, I heard many references to side projects or older projects which had been elevated to papers. The excitement they generated gave people the energy to work on scripting and modifications to a model code over the weekend. In that way, these informal, intense, and untargeted interactions were a form of doing scientific work and creating or modifying code products with minimal planning or structure, outside of that negotiated socially.

3.3.2 The Beautiful Chart

During the course of the study, he PI of the Omics-Lab group was concerned that the students and post-docs in her groups have to learn new programming skills, which she does not have time to learn. She was anxious about being unable to closely advise on that part of their work. She tells me this once as I pass her in a hallway after a talk, then she tells me with considerable enthusiasm that I should ask [student] about her "beautiful chart" that she made with R code. I follow up with the student, and she shows me the chart. It combines data she gathered with some data from the CustomInstrument-Lab team. The beautiful chart even prompted a post-doc in that team, George, to set aside a day that week in his busy schedule to "re-run samples," meaning to spend that day in the lab processing actual water samples to get her the numbers. Although it is easy for him, he had no reason previously to do it. The "beautiful chart" was this catalyst, and he was as excited as the PI of the Omics-Lab and the student.

As Mallory, another post-doctoral fellow in the CustomInstrument-Lab, tells me about a particularly fruitful conference where she was able to get a great deal of actionable feedback, I ask her a lot of questions about other people who use similar methods. She reiterates to me that it really is very few; that the ocean sciences in general are unlike the subset of "-omics" people: "the -omics people have more shared tools. Shared tools, diverse subjects. We have the opposite situation: a shared subject, but diverse tools!" Some people in the CustomInstrument-Lab do genomics, but Mallory is not one of them. I labelled the Omics-Lab with that pseudonym, though they, too, have to contend with a diversity of available tools. Colleen, a relatively recent post-doctoral addition to the group, comments: "I'm not an oceanographer, actually, I'm an environmental scientist. I came to the lab because I have similar background in some of the instruments that they use. I'm essentially using an instrument that I used in my Ph.D. [for freshwater systems] and applying it to [saltwater systems]."

At one of the casual lunch meetings with the BioGeoChem-Model team, people are bringing up the "clumpogram," or "hairball" chart made by Caleb in the BioGeoChem-Model group. Caleb is a new member of the group, a post-doctoral research fellow, and he brings a lot more observation and biology background. A diagram of his, which he showed in an talk where he introduced his thesis work, is referred to in jest as a "clumpogram" and a "hairball" during a lunchtime discussion of which visualizations to bring to a joint meeting between this group and a group focused on visualization from the computer science department. He is a little taken aback, a little defensive, but mostly he keeps asking for clarification: *which* of this charts is the clumpogram? In response, they quip: "oh, you *know* which one!" In his talk, he had indeed included multiple charts that had nodes and edges, and the edges were illuminated in an animation of the process of interconnectedness between nutrient availability and organism growth. These animated charts of biological process. By highlighting criticisms, they are articulating their own unfamiliarity with this visualization, but a desire to improve the visual communication and meet in a common ground. In Section 4.3, I will again return to the role of visualization to not only encapsulate findings and anchor feedback, but also to inspire action by way of being exciting, unexpected, or weird.

3.3.3 The Departure

The CustomInstrument-Lab, here, refers to a small team of people collaborating from oceanography, from computer science, and from eScience in order to take the best advantage of the exciting data generated by their innovative instrument. Although this instrument is used by a few others, it is not typically applied in this way: to do constant data-gathering while at sea. This produces enormous quantities of data, which introduces several challenges: monitoring data while at sea and on shore during/after a cruise; maintaining data for analysis and comparison across cruises; and archiving and presentation of data to the scientific community, funders, and stakeholdes. The entire dataset has to be "filtered." The filtering step is subject to revision. For example, through a comparison to some existing data, an ocean post-doc and an ocean [hardware] engineer realized their current filtering algorithm is

biased, so they began to working on another one, which will need to be applied retroactively to historical data once the lab tests (with the actual hardware instrument) validate a new approach. The analysis is typically at the level of individual cruises, either in R (favored by one post-doc) or iPython (favored by another post-doc).

The duration of the study was short enough that only a few people transitioned to the next professional position, departing from the teams. In the BioGeoChem-Lab group, one post-doc left, and two others were actively seeking, considering, and finally accepting academic positions. The software advisor had been working with the group at UW, and with the same PI at a different university where the group had previously been housed. In the RegionalNowcast-Model team, the software advisor who had been providing advice after-hours actually joined formally as a part-time staff person. The Omics-Lab only had access to various willing experts in the department who could be invited to a meeting or reached out to with a request, but in a different way than the in-house resources available to the other teams. But in the CustomInstrument-Lab, one of the software advisor staff from the non-oceanography side of the collaboration, David, left about half way through the study for another employment opportunity; his position was not replaced, but a different external was included, and the in-house S.A., Andrew, remained, but was already very busy (and very capable, but still busy). The following recounts some detail of the afternoon of his departure.

I had followed George to his office after a meeting. He was accompanied by David, a data scientist who had been working closely with both Mallory and George for over a year. In this conversation, David explained to George that he was leaving the university, having found other employment. George's reaction was threefold: to be supportive of David doing the right thing for his career; to be personally sad to lose him; and a panicked request to get David to explain things to him about how the analysis software works. As he asked questions, David sat on his knees (chair scarcity, and his own choice, as George did offer a chair) and frantically walked George through an issue he was having, in a kind, and really excited, teaching tone. Mallory walked in to see George and was a bit surprised to see

David there; George filled her in on the news. She is stunned, and then her first, jokingly furious question was: "will they write documentation [for the system]?" David says yes. She raises her voice, again jokingly: "give me NAMES!" Then, after David leaves, silence falls between George and Mallory. Then, they debrief: "This is a huge bummer for me." "My first reaction is selfish:" to be upset for the project, not happy that David is going to better directions. "David is going to be very hard to replace." "We spent a long time developing a relationship." David, who had initially worked with Mallory to get all the data into the database system, left in March. In Section 7.3, I tell the story of a data corruption that is exacerbated by his departure.

3.3.4 The Upgrade

For about half of the duration of my observation, the RegionalNowcast-Model team worked on switching from an older to a newer version of the model code that they have been using. Arather substantial upgrade demanded an extensive process of re-integrating the model itself into its surrounding analysis and data processing pipelines. The model is a widely used one. When I asked why they had chosen to use this particular model. It is a global model that they are modifying to work on a specific local region. The PI explains that it was included in the grant conditions, and jokingly says that it made it easier because she did not have to make a choice, and therefore had no need to compare options and justify the selection, "just go with it." The model code is created by a consortium, and has relatively poor documentation. Therefore, one of the services the group is doing for the broader community is publishing public documentation and tutorials that can be used by the numerous other users to this model code. As they are updating to the upgraded version, they encountered many "known unknown" challenges: they build in time and effort for parts of their analysis and web visualization pipeline breaking in the course of the upgrade, but the nature of the error and extent of necessary fix are unknown beforehand. So they are adding to the public documentation as they go through the upgrade. They are not involved in contributing to the code, only in building their own analysis around it, because to be involved in the model

consortium, the group would have to put in a certain number of hours per year into that development work, and they do not have that kind of resource.

In this team, there is an automated process that gathers data from other sources, like weather data, and combines it with model results to produce a series of visualizations for the website. Because the website is a volatile artifact that is continuously amended to reflect new analyses and ideas, the team's part-time Softwre Advisor, Ed, occasionally has to fix it. For example, once Ed and a graduate student added a new type of visualization to the flow, but there was a problem with permissions for the folder into which the images were being written. They had tested the code evening before at the end of the sprint. In order to get around a surprising and unnecessary challenge, they had changed the permissions and forgot to change them back when they were finished. This was by no means a critical error, it was diagnosed and addressed within a matter of an hour. Ed, along with the PI, are well-recognized within the rest of the team as keeping an eye on it after hours and on the weekends even when it is stable. Because the overall pipeline depends on scraping data from other sources, unreliability elsewhere cascades down. When the PI and the tech advisor are both out of town, a post-doctoral fellow (Erin) and a graduate student (Josh) are tasked with making sure it runs okay; throughout her first day on monitoring duty, she check on it frequently, mentioning during the meeting to other students and post-doc: "you guys will be happy to know the nowcasts[9] are still going. Let's keep our fingers crossed," to which one of the students replies: "when [Ed and the PI] were telling you what to do if something goes wrong, it was like telling the babysitter like ok, you're the big kid! heres what to do if things go wrong!"

3.4 Summary

This study is about **deliberate programming skill acquisition** in a field where a dizzying variety of research methods have burgeoned and co-existed, including inventing data collec-

[9]Very short-term forecast, e.g., a day out. In this case, the model nowcast can be compared to observations.

tion hardware, as well as decades of collaborative production on computational models and of sharing increasingly-complex global datasets. In this chapter, I provided a glimpse into some of the challenges and social dynamics I observed in the field setting of this study. In particular, I illustrated way in which oceanographers actively interrogate the validity of the scientific products affected by code work. I also introduced the role of the *software advisor* as a means for the study of social resources informed by computer science and software engineering. A software advisor (SA) may not self-identify as a "real programmer," but he or she is approached as an expert resource by others in the team.

In this chapter, I introduced the methods and the participant population. In Chapter 4, I will introduce the concepts of the working environment and the perfect world, and show how they are experienced by the scientists. In Chapter 5, I will expand the discussion to include the narratives, influences, and pressures of software engineering in the *programming-oriented* aspects of what is and what could be. In Chapter 6, I further expand the discussion to the full proposed model of change in scientific code work.

Chapter 4

THE WORKING ENVIRONMENT & THE PERFECT WORLD

In this chapter I develop two components of the proposed model: the *working environment* and the *perfect world*. My narration in this chapter takes on the language of the participants to offer the reader the meanings and values that inform the complex reality of "what is" and their collective imagination of "what could be." Visual display of data and results is a central component both of the expression and negotiation of this imagination. Chapter 5 will build upon this chapter to expand the vocabulary to include "best practices" as one form of influence from software engineering practice.

The *working environment* is that collection of tools, skills, contexts, and social arrangements in which a scientist produces code. Consider Erin, a post-doc in the RegionalNowcast-Model team, works with both observational data and models. She has two different working environments, or "set-ups" for the tasks, which are distinct enough in her mind that, in this observation session, she accidentally re-implements a small utility that exists in one environment when she is in the other environment. She forgets that she can use this minor technological resource across the environments, and this occurs to her once she explains to me what she had accomplished.

4.1 Personalized Working Environments

One of the post-SWC interviewees, a programmer with a science PhD in a high-performance computing (HPC) scientific context, lists his top 5 "data science tools and resources," in this order and phrasing:

"R and git first; then hosted version control service like GitHub and BitBucket; "obviously, StackOverflow is pretty important;" and "mostly markdown" or "a little bit of LaTeX", "because it's easier to put text on version control"" – SWC Interviewee P06

This list emphasizes specific tools as resources, including a programming language (R) as well as mark-up languages (markdown and LaTeX) as both part of a coherent coding environment. Another interviewee, also a software carpentry participant and a graduate student, additionally comments on her 5 top resources:

> I [would have preferred] Python or R instead of MATLAB or LabVIEW, which the lab is using. They are open source [rather than proprietary] and there's a lot of [examples online:] 'heres XYZ and you can just download this code and run it and it will produce what you want!' I use a lot of [examples] from StackOverflow. Wikipedia is where I go if I want to actually know what I am doing. ... Just the very basic computer science classes I took and then dropped in college [where I learned to] draw those flow charts of programming ... draw a logical chart to map out what you're doing. Then talking to other people, finding somebody that knows how to do what I want to do in whatever language and then I'll just learn that language or that part of that language so that I can do the thing that I want to do. – SWC interviewee P12

In the above quote, there are the familiar on-demand on-line resources (StackOverflow - mentioned by everyone; Wikipedia - mentioned by many). The interviewee P12, quoted above, in addition to specific tools, cites *drawing* as a resource, and *asking colleagues*, which below are categorized as cognitive and social resources respectively. Additionally, the interviewee articulates a misalignment between her preferences and her working environment, which is dictated in part by the lab. Another SWC interviewee going into his first faculty position mentions a similar tension between his preferences and his prior context, which then inform his choices of establishing best practices in his own lab[1]. In the first quotation, from

[1]SWC Interviewee P02, quoted on the subject in the beginning of Section 7.2

SWC Interviewee P06, the participant elaborates later that he does not work closely on code with anyone else. This allows him to be "probably faster, because I'm never waiting on somebody else to finish anything [but also a little bit dangerous though because I have no checks." Aside from lacking additional external scrutiny, P06 is able to be more free with his choice of tooling.

Working environments embody personal preferences and context-specific requirements in the programs and interfaces (scripting, GUI, etc.) that are chosen from from many available and competing options, for code work. I asked participants in the Software Carpentry workshop (see Figure 3.1 for study participant population breakdown) about their favorite resources, online and offline, used in the course of coding practices[2]. By asking participants rank the resources, I pushed them to think beyond commonly-cited tools and to generate ideas on how they define "resource". The protocol for these interviews (included in Section 3.1) was purposely open-ended to allow a range of answers. "Resources" include not only specific programs or websites but also their problem-solving approaches as they are supported by specific technologies, or mediated by social interactions.

Three of the most universally commonly-cited resources were **on-line on-demand** resources; **high-maintenance social** resources; and **guiding** resources. On-line, on-demand resources include StackOverflow and Google search. In the sense that they are produced by other users on communication platforms, they are "social" resources, however they are different from the high-maintenance resources of emailing an expert colleague for help, as one participant describes: "If I have people that are good with a certain software I'll tap them on the shoulder and say, 'Hey, do you know how to do this?' or send them an email as I go through. The so-called "high-maintenance" resources may serve secondary functions, opening doors for different conservations and creating awareness. Nevertheless, it stands in contrast relative to the vision of software as uniquely easy to share (as in[30]). I usually

[2]The phrasing depended on how the participant had described their own work: their "data science workflow," just "data processing," or "writing code" - with my interests being consistently on the amorphous but omnipresent collection of code work activities and practices

tailor my questions to specific people, and I'll pull a couple of those issues out of my list and send them on so that I don't overwhelm my contacts."

The "list" that the above quote mentions is a particular *cognitive resource* that was also not uncommon: "I have an issues list that I might have time to get to." The "issues" include "little things" to get back to, "dirty shortcuts that worked" waiting to be cleaned up, and more generally "speed bumps I've been hitting in software." This can be seen as a *guiding resource* aimed at a "future self," to use a phrase[3] from the SA of the RegionalNowcast-Model team. The SWC interviewee continues: "I try and make a notes list of quick shortcuts that I don't use very often but will find handy and might not remember in the future. Sometimes my [notes on challenges and problems] turn into quick shortcut cheat sheets." Other guiding resources include tutorials that combine actionable things (e.g., code snippets) with explanation. Those that were noted as especially useful by interviewees were ones put together by people in the field who had created the tool or used it extensively and were seen as relatable or recognizable individuals. For example, Jake VanderPlas, a member of the UW eScience community and Assistant Professor at UW's Information School, was mentioned offhand in a variety of contexts during this study, and with praise for the quality of resources he had produced.

Especially effective note-taking techniques and tutorials become staple cognitive resources in the working environment. In many cases of deliberate skill-acquisition or learning studied, there was a sense that the purpose of learning is to expand one's imagination by way of growing the vocabulary of what-could-be, as a way to search out new technical and social resources (which are represented, respectively, in the emphases added to this quote from an SWC interview participant):

> Half of the importance is being introduced to a subject, especially one that I have hardly any training in. It's basically how to ask intelligible questions; there's going to be things that [a workshop] doesn't address, but **being able to ask an intelli-**

[3] "Be kind to your future self," said to a group of learners, on why they should document early and well.

> **gent question** and **understand who might know the answer** is half battle.... One of the reasons why I love Wikipedia because it at least tells me what terms I should Google. – SWC Interviewee

Differences in working environment arise not only from the passage of time, but from the personal differences between individual working environments. All code is subject to "bit rot": the degradation of code that is not touched over time. This degradation is the result of inevitable and ongoing changes to the working environment, or of upgrades to various components on which it depends. This creates an additional barrier to collaborating directly on a figure. In one case, a post-doctoral researcher was working with a graduate student over a distance on a manuscript. Although all analysis code was in GitHub, and despite extensive exchange of figure code, he told me that they still could not produce the same figure. The trouble was that he was improving the analysis code, which made for two alternative versions of it: the one "frozen" in time and the one under ongoing modifications. The student used the one in flux, rather than the one frozen in time, which generated one form of miscommunication resulting from trying to reproduce a figure in a different environment. In another case, the student has made an incorrect assumption in her code that resulted in a chart that looked like nothing really interesting was present in the data. When the post-doctoral researcher identified and removed the error, the resulting figure was amazing, exciting, and clearly demonstrating a strong correlation.

The social, cognitive, and technical components of an individual's working environment are all interdependent. When Erin first joined the RegionalNowcast-Model group, she was required, like everyone else, to transition to using version control (hg and bitbucket) in a way prescribed by the group: folders for different researchers, plus documentation on a separate site dedicated to creating public documentation as an appreciated service to the broader oceanographic community. The documentation work is associated with its own working environment, spanning all three types of resources. The documentation site itself constitutes a technical resource. This technical resource becomes a cognitive resource when

we consider not the implementation, but how it is a "living document" instrumental in supporting collaboration within the group and introducing new members to it. Its sustained existence as a "living document" is supported by the social resources: regular, nearly-weekly meeting where questions of what documentation should go where can be answered.

Josh, a graduate student in the Regionalnowcast-Model group, told me how the practices and standards in the lab, which we can interpret using the working environment vocabulary, has changed in the past three years. The CustomInsturment-Lab also had a regular meeting where team and collaboration members were able to bring their questions about where documentation or data files ought to go, or to announce in-person some change in the library they were writing or an improvement in its functionality. However these were less frequent and also spanned many other concerns that tended to fill the space of the meeting much more frequently, such as paper publication venues and plans that make the most sense for recognizing the contribution of different members of the interdisciplinary project. This meant that, in practice, the technical/cognitive resource of particular commented sections of code therefore extended to those few people who actually wrote them.

The utility of the *working environment* as a conceptual tool is to allow us a greater degree of freedom for talking about the adoption and adaptation of new tools or skills. For example, Mallory, a post-doctoral researcher in the CustomInstrument-Lab group, explains to me why she is "investing" time in re-making her visualization in Python rather than MATLAB. Her MATLAB license had expired, and so this was the trigger to make a decision, at which point she decided in favor of the open-source, free technical components of her working environment. Furthermore, she describes code for visualization as both (1) persisting for each published visualization, to aid documentation reproducibility; and (2) extensible for similar visualizations later on, or at least modifiable in terms of formatting. The pieces of code in question act as both technical and cognitive resources, respectively. As another example, a "sequence database" may not be a database, in the sense of a structured way to store data that abstracts interacting with its semantics away from the details of its storage

on disk. A text file with sequences may constitute a usable database, as the text-file fulfils its technical resource requirement.

The working environment includes visual representations as a central cognitive resource in the oceanography setting. Operations within the environment are inspired by wanting or seeing a chart or graph that has "weird" or "interesting" features. Spotting these relies on the cognitive resource of a scientist's training, focus, and research taste. An example social resource in this scenario is a discussion that resolves these features into a causal explanation. The result of the discussion is either an explanation of how the chart "makes sense" or a concrete next step to investigate the "weird" feature or further explore the "interesting" one. The level of scrutiny applied to both is similar, though the "interesting" receives enthusiasm where the "weird" receives a persistent, puzzle-solving sort of dedication.

Because of the social nature of this questioning, the technical resource requires shareability. However, it is the technical resource *as a whole* that has this requirement, not a specific part of it. Gathering around a monitor, for example, is the most common. This imposes no requirements on the specific technologies used to produce a picture. Other options include printing and laying out the charts and projecting during a group meeting. The core design requirements here are ease, control, and reconfigurability. The advantage of the printouts, for example, is their affordance to be rearranged into different pairs or sets. It is very rare for a single chart to be the subject of attention. Typically, multiple charts are discussed, either in context of drawing a comparison or in triangulating a process or feature of interest across multiple slices through space or time. In the case of projecting charts, either an interactive iPython Notebook (iPyNB) environment or a presentation slide-deck allows moving between multiple charts of interest, created by that individual in an informed anticipation of the discussion. In the case of standing around a computer, the programming environment which produced the charts in the first place (R, MATLAB) is at hand to help generate new images on demand as part of a process of rapid exploratory analysis.

Comparison and triangulation, which is at the core of the exercise described above, can be done either at the level of plotting additional data points for comparison, or by articulating

a known pattern (for example, "we expect to see this kind of salinity at the surface, but not in the deep"). This "informal" comparison that draws on shared knowledge - articulated as a hunch or as a whiteboard squiggle - is a crucial step of interpreting and interrogating a new data artifact. In terms of tooling, any visualization environment which encodes too much of the comparison violates this requirement.

The resources comprising the working environment are not static but subject to evolving capabilities and constraints. For example, an SWC Interviewee notes that: "I suspect there will be a lot of changes, but for the positive... once I start working on the statistics part and the automation integration with two other researchers on my team, we have to have protocols set up, like how to create meta data, how to format scripts in R ... their structure to create the meta data script type thing, and ... an R add on for web data visualization." Needing to "have protocols set up" articulates a need for social resources, and the cognitive resource of data visualizations is recognized, however the primary way of articulating the working environment is through the listing of various particular packages and associated protocols. In the next section, I describe the collective imagination of the perfect world, which is that implicit possible future against which the coming changes can be interpreted as being "for the positive."

In the first few weeks of my observation, and of his membership in the BioGeoChem-Model team, Ryan gives a lengthy talk recapping his prior dissertation work. In Ryan's talk, we witness an exchange of tooling approaches, enabled by Ryan's methodology being relatively novel to this team. In his "overview slide" he mentions microbe counting, which surprises the PI because he would have expected it to be using image processing and be more "advanced." Although the "manual" process of Ryan's unfamiliar method is vaguely disappointing/disillusioning, the relatively more familiar but still "other" process of PIB's sequencing (she sends samples off for sequencing and is getting rid of all her own sequencers) is amazing to the PI of the Omics-Lab group: she literally says, "it's amazing! We are 10 years out. I am shocked." Another member of the group points out that their in-house

sample analysis process also allows additional specialized contexts. This is necessary for the kind of methods development research this group does.

During Ryan's talk, the PI of the team points out, as he typically does, ways in which the findings could be delivered more effectively. The timbre of this feedback, which is common in the practice and casual talks in the team, is about showcasing the work. As one of the students puts it, "the audience deserve to be wowed" by a particularly impressive chart, so it should be given more time in the talk. In this case, the PI notes that "a model would be a nice way to show what you're talking about," while one group member asks whether PCA (principal component analysis) might be useful, and another asks about possible applications of machine learning. These comments reflect their respective current methodological interests, as I discover later in follow-up casual conversation.

Other people's approaches can pique curiosity, as above, but also inspire awe or judgment. For example, Mallory is presenting by way of scrolling through her iPython Notebook. In the course of narrating her findings, she casually and quickly implements a small analysis suggestion from the audience on the projector screen. This impresses the PI of the CustomInstrument-Lab team: "you just did that *on the fly*?! *Really cool!*" On another occasion, a graduate student and the PI from the Omic-Lab team are discussing another group's cruise data collection protocol and set-up. They express amazement by the choices in the protocol, which would make it unnecessarily and unreasonably slow, in their opinion.

Continuing his answer about resources, Software Carpentry participant interviewee P06 comments on looking at other scientist and programmer blogs about their work ("Also, I follow a number of different scientists on Twitter. If I notice a tweet that somebody that I know does pretty good work and seems to produce a lot of impressive figures I check out their post and see if there's anything I can glean from that"), especially those supplemented with interactive visualizations ("a lot of people use IPython Notebooks these days, which is very nice"), saying that he "looks for a little bit of both [inspiration and examples]. I look for examples of realized visualizations that appeal to me [that may] appeal to other people in my field. I also look for examples where people have done things in kind of a cool way

that I previously haven't thought about, or didn't know what was possible. It expands my horizons of what I can think of in terms of what I might try in the future." This interviewee, who does modeling work, has integrated into his working environment a resource that allows him to continually expand his imagination with "cool" new approaches.

4.2 The Perfect World

Ways of describing the shortcomings of the working environment are strong signals for ways in which improvement in process is conceived and pursued. A software advisor in the CustomInstrument-Lab team commented: "in a perfect world, there would be a satellite for data transfer during cruise, but this is not that world, so data is transferred once in Hawaii." The software advisor of the BioGeoChem-Model group commentd, during almost every explanation of the current working environment, that it is "not ideal" while providing a historic or domain-specific reason for it being so. Acknowledging imperfection in the current working environment can be seen as a way of comparing it to an imagined "perfect" world (what-could-be), and founding it lacking in a way that informs learning. Drawing attention to the imperfections is not quite defensiveness, but ubiquitous comments like describing something as "kind of terrible but I still think it's really useful." The word "perfect" was chosen because of its relative ubiquity in data, along with "ideal," used similarly in saying something is "not ideal" or in saying that "it would be ideal to ...". In other words, the "perfect world" is that world which you would re-build, in place of the current, "not ideal" reality, if only time were to stop, and you to have the opportunity to apply everything you have learned from the first time around. In this section, I elaborate on the uses of the phrase "in a perfect world" and the descriptor "not ideal" when describing the current solution. First, I introduce the "collective imagination of a perfect world" as a social practice. Second, I describe some of the features of the "perfect world" when it pertains to code, which are illustrated in the Figure 4.1.

Figure 4.1 includes not only the desirable characteristics of code in the perfect world, but also ways in which they can affect different audiences. All code work has *some* audience,

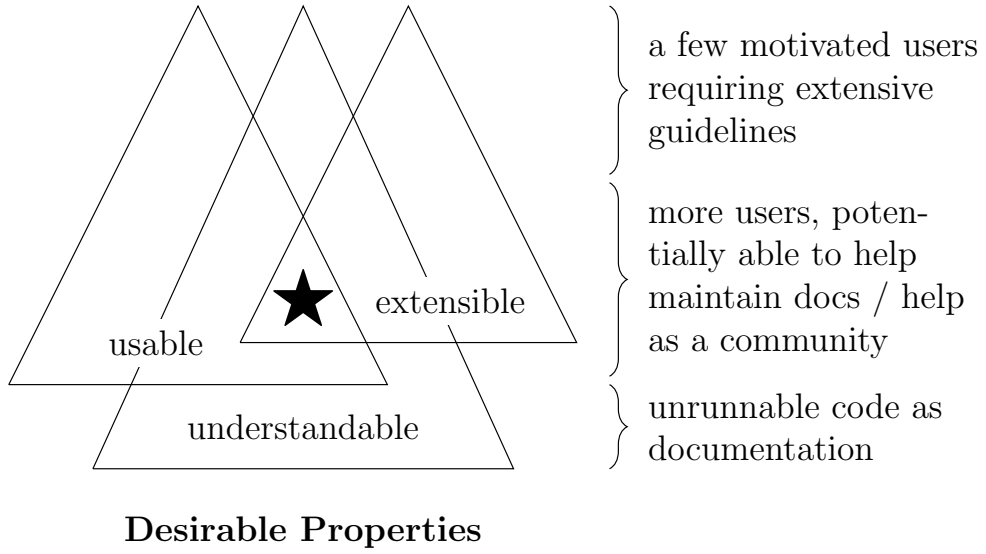

Figure 4.1: **Desirable qualities of code in the *perfect world*, as they influence audience access.** In the perfect world, denoted in this illustration with a star, code has the overlapping qualities of being usable, extensible, and understandable by everyone. Something can be "understandable" without being usable or extensible if it serves to document or guide. For example, code on StackOverflow sometimes cannot be run, but is still "understandable" enough to be useful. Code cannot be both "usable" (in an end-user sense) and "extensible" (by a motivated coder) without also being understandable. Scientific code typically resides at the top of this chart: it can be made to have any of the properties, but only to motivated uses through extensive, contextualized guidance. Moving downward in this illustration toward this star expands the potential audience, reduces the amount of "hand-holding" (see Section 7.3), and comes closer to the perfect world.

at the very least (1) the future self of the programmer, or the current skeptical self of the programmer. Additionally, the potential audience of code work includes (2) lab members, colleagues, and others who have a relatively high familiarity with the work and are exposed to intermediate version, help with various challenges; (3A) colleagues working across time: from prior projects or workplaces, or hypothetical future colleagues or students; (3B) colleagues working across space: from other institutions or labs, where there is mutual interest in sharing data (in either or both directions). Because the working environment is a complex and personal collection of inter-dependent, highly contextual parts that undergo continuous change, *all* of these audiences (including the future self) require additional translation, integration, and verification across distinct environments.

Comparisons to the "perfect world" constitute a future-oriented way to identify the perceived deficiencies of the working environment or its products. These are in contrast to past-oriented critiques: the explanation of something for "historical reasons," or for reasons that involve identifying a long chain of stakeholders or programmers, their contexts, and an explicit recognition of how this has resulted in the current "not ideal" situation. The *expertise* necessary for the "perfect world" comes through in the precision of statements about it. The precision is informed by the experience of specific disappointments: it is a retroactive, "hindsight is 20/20" kind of expertise. The difficulty of attaining perfection is that the path forward remains unclear despite lessons learned in the past. When articulated as a "specification," the perfect world is imagined as something that could realistically be implemented, but only if there is someone whose primary role includes doing so. This sentiment is expressed both by people whose roles only peripherally involve programming as well as by full-time programmers focusing on technology. Regardless of how attainable the perfect world is said to be, it is not actually attained. It changes in response to what *is* attained, and every role is filled by a person who feels stretched to the limit of their resources.

I distinguish the imagination of the "perfect world," which requires some expertise and familiarity, from the imagination of the "silver bullet." At one point, the Omics-Lab group

decides to try out Tableau[4], with a great deal of initial excitement, followed by a bit of disappointment: this is not the solution to everything, though it does have its advantages for the group in particular contexts. When a post-doc first tries it out, the PI and the group are excited, which shows in an email exchange about the lab meeting (it goes from the PI asking if the meeting is necessary and whether it ought be rescheduled, the post-doc replies with an excited message about having charts to show, and the meeting is enthusiastically back on).

The PI of the group was direct and explicit about her anxiety over new tools/skills (and perhaps not using them enough, or not knowing the options well enough) and wanting to do something, and thus seizing on the opportunity of my showing something very simple to the post-doc. It is not mind-blowing and what she makes with it is not perfect. Later, another group member, a graduate student, uses Tableau to make some plots, switches to R because Tableau does not offer enough control over formatting, and produces "beautiful" charts that are the subject of great excitement, as the data is convincing and expressive.

Excitement about the initial attempt to use Tableau was revealed through emails prior to the weekly meeting, in which the post-doc wrote that she had Tableau charts to show, and to which the PI replied with enthusiasm. The exchange involved exclamation points and happened in the morning before the noon-time meeting; not surprising for a friendly group with robust meeting scheduling practices, but unusual relative to other meetings I had attended[5]. Excitement about the second "beautiful chart" was expressed in a mention in the stairwell from the PI that I should "ask Alice about her beautiful chart! She made it in R!"

When I followed up with Alice, a senior graduate student, she told me that Tableau did not offer the fine-grained control over visual aspects of the chart that would be necessary for it to be a publication-ready visual artifact. She then walked me through the different-size

---

[4]GUI-based desktop data analysis platform that leverages relational data models and novel visualization techniques to provide a visualization-centric alternative to spreadsheets. This group had been struggling with the kind of Excel challenges that are used as a motivating example for switching to Tableau, so it seemed like a natural fit.

[5]It's possible some of this excitement was for my sake, but I strongly doubt all of it was.

and different-colored disks arranged to follow the path of the cruise on a map of the region of interest, which she made in R. One of the visualizations was a series of pie-charts arranged in a table and varying by size and composition, which is more reminiscent of the Tableau visual vocabulary than that in oceanography talks. I even asked about the inspiration behind the choice of visual mapping, and Alice confirmed that she had re-implemented the Tableau variant in R. In an indirect way, it had an impact. However, even if it did not, it needed to be tried in order to be rejected for being insufficiently controllable.

On a different occasion, the software advisor of the RegionalNowcast-Model explains to me how many of their Python scripts use a large model code with many constituent packages and many diverse users internationally. He describes to me running a single run with a single command. In "a perfect world," this hypothetical command would deal with creating symbolic links, creating directories, and adding jobs to the queue manager in the computing resource. Output in this world is one output file per variable per process, (e.g. on order of 2K files that need to be re-integrated into a more coherent result). Several months later, he unveils such a perfect-world command to the four students and post-docs who comprise its current users. The demo is riddled with mentions of the many ways in which the command would be improved in the perfect world 2.0. The programming and documentation work that went into this project had as its audience the current researchers, as well as the unknown but expected and desired new graduate-student additions to the group. The previous perfect world of having a single command had expanded to having not only more features, but also to a wider and more diverse audience.

A post-doc in the RegionalNowcast-Model group explains to a junior student, as she sits doing a backseat-advising kind of pair-programming: "In an ideal world, our code runs perfectly and the user never looks at our code, but thats not what happens in practice. In practice, there is always some case we havent accounted for. So, first thing - will someone be able to look at this code and understand? I like all the docstrings[6] [she points out what

[6]In python, a way of commenting in a function that allows that comment to appear as API documentation later when the functions are used/ auto-completed as the programmer types.

the student has done well before offering a suggestion] but would be nice to include some inline comments." In this case, docstrings will provide verbose descriptions on space/tab-complete for users of the Python module. Inline comments will be helpful for someone who is examining the code itself, rather than only using it as a working black box. Both of these are ways in which the researchers are enabling the code to communicate with its various audiences.

The perfect world is not just a property of a technical component, but of an overall system that may have technical components embedded in it. Talking about collecting data in the sea, a member of the BioGeoChem-Model group is talking to a colleague whose data he is considering using for a very specific research task that involves making a comparison between their datasets. They are discussing measurements made by profilers, and there is some discussion of whether the profilers follow eddies. The colleague explains that because the profilers go so deep, they are basically random with respect to the eddies. The post-doc exclaims, "that's perfect!!" because it means the sample can be comparable. More often than not, the in-depth discussion of data applicability yields the verdict of a less-than-perfect-world. A graduate student in the Omics-Lab comments, regarding using the mass spec for chemical analysis: "Then in a perfect world, you have standards for all of the compounds that you're looking for that you can compare with and say this is how much standard I had, and it gave me this big of a peak. This is how much of the peak was my sample. Then I know how much was in my sample."

Because it is situated relative to an audience - even if it is just the future self, which tends to, upon questioning, also imply hypothetical future collaborators - the perfect world is *collectively imagined* in that it reflects values that are shared among the people to whom the complaint or the dream is articulated. In the CustomInstrument-Lab group, semi-joking frustration also reveals an exasperation with computer scientists who try to hide the inner workings: "we are *scientists*! looking inside and figuring it out is what we do!" In the BioGeoChem-Model team, I also heard joking about how hard it is to get the software advisor to explain what he does: "he doesn't like talking, just fixes all our problems super

fast." In both cases, there is a need to understand not only at the practical level, but also in terms of seeking understanding as a core aspect of the craft of doing science.

Understanding is at the heart of the perfect world, and can be in some ways incompatible with usability. Ryan tells me, over a lunchtime discussion, of the difference between two major programs, also reflecting that more-control-is-better but in a different tune. He compares and contrasts the two programs by saying that both are "goliath programs" and one is more "conservative" (in terms of including new packages) which makes it generally more "reliable." Furthermore, neither is especially "user-friendly" because the "programmers don't want people to get too complacent." He tells me about the goliath packages in context of having asked me to comment about my research so far[7]. I commented on how my study was less about collaboration between oceanographers and computer scientists, with the scientists primarily playing the role of the user, and more about code work that is done *by* oceanographers, which includes working with modeling code as well as small scripts[8]. Though my intention was to make a neutral statement, both Ryan and Mary corrected me with one of the many "not a lot of people do what we do" that I would encounter through the course of this research. They told me that great deal of researchers - those not included in this study with an inclusion criterion of active engagement with programming education or collaboration with computer scientists - used such large "goliath" programs. Though upon explanation, it was revealed that this too would require at least a little "kleenex" code.

However, there is a great deal of code that is neither "kleenex", nor "library", nor the other categories suggested in Fig. 3.2. In a way, it is not even code, but a highly particular "figure caption" or "experiment annotation" that is pervasive and produces a lot of code products, but which does not exist in isolation from other code practice. The scientist who

---

[7]For this situation, my intention was to reply truthfully about the study but also not give away "too much." Especially in this group, which was wonderfully friendly and inviting to me, so my main concern was that if I gave too much detail of what I was expecting or hoping to find, I would be shown whatever it sounded like I wanted to see.

[8]these, as opposed to modeling, are grouped under "Kleenex code" by Heaton and Carver - hence the title of the section [27]

has written nothing but figure code in a few months is likely to have written a considerable portion, or the totality, of an analysis or pre-processing script or program in the past, or likely will write it in the future. On the other hand, the "kleenex" code for analysis, processing, or figure-generation can be both a cognitive resource, a detailed lab-notebook protocol of what was done, and a reusable technical resource (reusable). The cognitive resource is effectively persistent when it is readable by scientific peers (or by the future self of the original researcher) enough to be able to replicate the action; it less important that it "work"and more important that it remain accessible over time. The technical resource, on the other hand, is effectively persistent when others are able to run it, and to modify it as necessary. It still must "make sense," but also "work."

Although this type of "code to document a visualization" is useful over time and a core part of scientific work, it elicits off-hand comments from some informants that they "shouldn't be copy-pasting." This reflects a larger "best practice rule of thumb" that the moment a programmer begins to copy paste code is the moment she should re-evaluate her actions and switch to a more modular design (this is the tone of the best practices in [70]). In the context of "traditional programming," "persistent code" is synonymous with "implementation that persists overtime while its context of execution changes," which means this code is subject to degradation relative to its slowly-evolving specification. In the context of scientific programming, "persistent code" can *also* be code that is subject to no changes in execution context.

The tooling challenge here is to capture this back and forth in a way that can be accessible later. On several occasions, the chart or data of interest was produced by those retired or dead, therefore unable to engage in the back and forth, which limited the interpretation of what was in the papers. A record of questions asked by other researchers and answers is as useful as the answers. After all, StackOverflow addresses a vital and universal need. Nevertheless, doubtless other researchers in the past had asked some clarifying questions, and access to those may prove a useful history.

Another distinct but pervasive social interaction over visualizations is again focused on the individual corralling the data and model run results from others in order to generate a complex visualization with many comparisons. Even if the data or results in question are published, there is still an email chain to discuss those assumptions and considerations relevant to the research question but left out of the manuscript. At least as much time is spent procuring this documentation, in the form of emails and Skype calls, as on discussing which details to leave out of the manuscript or the supplement at the time of publication. Asking questions about charts and data is a kind of expert knowledge transfer, as well as a daily recognition of ownership, arising from an expectation of a particular investigator being the most well-versed person in their particular area.

Code written in this kind of professional exchange of annotated visualizations is itself the documentation. In a working environment that spans not only technical resources but also cognitive and social, the visual vocabulary as it is used to communicate constitutes a *cognitive* one that further serves to engage with the *social* one. The code that operationalizes this vocabulary is stand-alone, persistent, and essentially non-modularizable: the goal is to "freeze" the visualization in time until it requires amendment in response to reviewers. Visualizations that are relatively more complex (in terms of including multiple sources of data or representing a multi-part analysis), even if not necessarily intended for publication, still remain the best workable summary of the experiment. They encode not only the question and answer, but the provenance. The worst case is the folder full of figures whose origin is unknown, especially if one of them looks intriguing. The best case has been one-off iPython notebooks that have gone through a process of being "cleaned up" and annotated with "figure captions."

Code to document a visualization was introduced previously in this section in a story about Mallory re-implementing MATLAB visualizations in Python. This kind of code is not subject to internal degradation through shifting execution context, it is still subject to external degradation of code: the libraries it depends on, the compiler that reduces it to machine code, the programming language specification may all change in the time between

when the "figure caption" was frozen and the time it needs to be unfrozen. When faced with this variant of "bit rot," Mallory noted that at first it was frustrating to have to re-create all the charts (previously in MATLAB) in Python, but this was a preferable time investment over updating the .m scripts, because she had been "meaning to learn" to make publication-level charts in python. The need to spend several days polishing python code was, for her, enjoyable and rewarding: it was relatively straightforward, and it resulted in a skill that she was happy to have honed. She used playful, constitutive[9] code work in order to make a technical resource with longer-lived usefulness.

Visualization code becomes social not only in discussion over a single artifact produced and managed by an individual, but also in an iterative work by a small number of individuals who all have stakes in the visual artifact and contribute to its formation. Consistent with the prior case, the potential end result for a chart is inclusion in a manuscript as a figure, so retaining fine control over visual elements like colors, styles, and fonts remains paramount. However, if multiple people edit the same visualization, they must balance the need for control with the challenge of using low-level scripting. Lastly, recognizing the different roles that code might play (e.g., communication, rather than automation, as the code used in StackOverflow discussions) can help distinguish *which* code ought to be subject to which evaluation relative to the best practices of the perfect world.

[TK This is a new paragraph added post-distribution. Return to this.]Although I spend most of this section talking about the vision of the perfect world in terms of the vision of how programming can change work practices, the collective imagination is not exclusive to code work. As an example of a collective vision of a perfect world which involves code work but which is not, at its core, programming-oriented, consider *reanalysis*, on which Edwards spends a chapter in *A Vast Machine* which is on climate modeling that includes ocean general circulation models. This is an algorithmic approach to combine historical and model data, which "has been disappointing ... as a replacement for traditional climate data" [18,

[9] not mutually exclusive

Chapter 12]. Aside from "significant differences... between the major reanalyses as well as the reanalyses and major traditional climate data sets," the issue with reanalysis is that although the results agree reasonably well for "variables constrained directly by observations, such as temperature," not for the derived variables: "for example, reanalysis models do not yet correctly balance precipitation and evaporation over land and oceans, whose total quantity should be conserved. This affects their calculations of rainfall distribution, a climate variable that is extremely important to human populations and to natural ecosystems." My inclusion of this passage is not intended to highlight reanalysis and its relative successes alone, but to highlight the extent to which particular technologies (modeling) allow actions that are not merely hard but impossible (calculating derived variables over long time-spans) so long as there are means for verification (conservation laws and observational data constraints). In the case of reanalysis, the hope of the technique did not match its "disappointing" reality despite the fact that "the do-over has already been done over several times," with the approach being "hugely expensive, requiring many weeks of supercomputer time as well as ongoing data cleanup."

4.3 Visualization and Storytelling

Code that works settles into shared libraries or infrastructure and becomes invisible, apparent only at points of breakdown or through contingency planning [59]. The code is also visible during the constructive phase. At the moment that a piece of written code is executed, it becomes a kind of autonomous agent. All code observed in this study, spanning scripts, models, and analysis pipelines, was intended to be understandable, but took iterative improvements to get there. During that time, the "agency" of code consisted of those actions which lay beyond the vision of the programmer by virtue of unexpected side-effects or bugs. In an analysis of 200,000 bug report titles, Ko et al. report that "95% of noun phrases referred to visible software entities, physical devices, or user actions" [33]. Even functional and at some level understandable, some code (e.g., complex Intergovernmental Panel on Climate Change (IPCC) general circulation models (GCMs)) can be oblique enough to demand the

treatment of itself as an agent, such as when as the researcher "figures out what the code is doing."

All code acts on the world according to its own principles, which are intended to be understood by the researcher, but in practice diverge in inevitably surprising ways in the actual implementation. As such an agent, it inspires hope, scrutiny, playfulness, and fear. In reference to discovery and exploration of available tools and processes, the phrase "playing around with [e.g., a new tool or programming language]" is at least as common as "learning about [same]." In describing a problem, a scientist may say that a program "does not like" particular values or actions. In other words, the program is buggy, but the "animating" narration imbues code with the capacity to petulantly reject a request. In particular back-up choices, the coding scientist may reveal fear of irreparable damage - by changing the code or by unleashing some big automated operation onto some precious data. The code may or may not "work," and the scientist concerns herself with ways to "convince" herself that it works. When it *does* work, the program has the capacity, by virtue of speed and complexity of the actions it can carry out relative to a human, to awe and delight.

When code manages to inspire joy even relief, as opposed to frustration, it does so through the charts and figures that is it able to produce or facilitate. Visualization is not only ubiquitous, but elicits so much interaction that it serves as a primary way for an observer (me) to be able to witness participants reflect on their work practices. While bringing me up to speed on her work following a conference, Mallory jokes that "[we are] professional skeptics in science, and sometimes [it is frustrating; and you want to say:] 'just *shush*, I'm just showing a figure!"' Throughout the study, it became apparent that most of the time, there no such thing as "*just* a figure," confirming Mallory's quip about professional skepticism toward every figure. Visualizations are central to the scientific practice observed; they are routinely used to interrogate the analysis process in the course of an individual doing trouble-shooting, as well as in a collaborative feedback setting. If a visual display of data or information is shown, on a computer screen or the projector, it remains there for at least a few minutes as those present consider it and ask questions and offer their ideas.

How do these visual objects support narration and interrogation of research process? Storytelling using visual artifacts "animates" the salient objects - oceanic ("the little guys"), chemical/physical ("the particles"), or technological ("the function") - as primary actors in stories, imbued with actions, desires, and preferences. For example, "this function *wants* this type of thing for input, and it *doesn't like* for the input to have such-and-such properties." Animating these inanimate objects focuses scrutiny on that object during trouble-shooting that traces the provenance of data as is is carried from the water, through data collection hardware or filtering protocol, converted into lossy numerical representations, and then transformed to match other data as part of triangulating some conceptualization. Code written to facilitate these transformations must be transparent, understandable, and convincing. When a study participant says, "I am just tyring to convince myself that it works," she is applying the same level of scrutiny as she would imagine a colleague would in a collaborative session or during peer review. If an interactive chart is not persuasive in a manuscript, then its utility in the course of analysis is diminished relative to a chart that could be used to build an argument to scientific peers.

Narration of physical, chemical, or biological processes is built up from first principles, taking into account the connections between different resolutions, and ultimately is intended to "make sense" in an intuitive way to the trained ear. For example, if a hypothesis involves adding a lot of freshwater to the ocean, one way for this idea to "make sense" is by doing a back-of-the-envelope calculation of how much fresh water that would need to be. If it exceeds by an order of magnitude the effect of the ice caps spontaneously melting, that is just too much for the behavior of the model to apply to our present reality under conceivable conditions. In telling the history of climate modeling, Edwards in *A Vast Machine* describes ways in which climate and sensor data is created by the instruments: the digitization and instrumentation processes themselves modifying the resulting numbers and how they can be used. This constraint the freedom to interpret on corresponding data provenance is evident throughout the collegial debates observed, but the complexity of the provenance challenge is reduced to the indictment of numbers as "real," "made up," or "guesses."

For example, Colleen, a post-doc in the Omics-Lab team, is showing some new visualizations she had just made in Tableau, where she has plotted concentration in *fmoles*. Femtomole (fmol) is $10^{-15}th$ of a mole. It is a unit for measuring the amount of a substance. The question is, "how much of *this* thing is in *there*?" The answer might involve counting, if the thing is organisms, and the goal is to study their death or reproduction. The answer might be in grams, if it is a nutrient that is used up. A mole is a counting measure used in chemistry to reason about the quantities of molecules as they react to form other molecules. In the equation $2H_2 + O_2 \rightarrow 2H_2O$, it would be confusing to measure in weight, since the molecular weights are so distinct. The PI of cuts her off (in a welcome, teaching, but uncompromising manner): "fmols are not real." In her typically somewhat playful and casual tone, Colleen retorts that the units are "...*sorta* real?" In this case, the PI does not shift toward neutral pluralism that I have pointed out before[10], but instead explains that the distinction between counting and weight produces "totally different shapes" for two outputs that are bout "about amount," even though both are at least "sorta" real.

The visual language of this setting helps not only convey the information, but some sense of how this information was produced, at least at a birds-eye view. For example, below I explain discrete marks and color gradients in a few typical variations as immediately signalling something about data provenance, as part of a consistent visual debugging language.

Discrete Marks. Multiple dots can refer either to observations in the environment or to "different worlds" produced by a model. Different model runs, parameterized in a different way, are comparable and therefore can exist on the same plot. Marks can be dots, or more complex glyphs such as ellipses, to represent current magnitude and direction in a single, accepted mark. When presented in a spatial plot (with X-Y axes mapping to longitude), the marks refer to measurements in that area, and may be glyphs - using color and shape to encode additional variables. When there are lines as well, the lines correspond to continuous

[10]See the end of Section 3.2.

model results (over time or at varying depths) that can be compared to the discrete dots of measurements.

Bounded color gradients. Edges delineate the region of interest by bounding the bottom of the ocean, and the banks separating the shore and the sea. If edges of a particular region are shown, the region of interest is typically also overlayed with colors. Here, the rainbow spectrum might be used, as might the blue-white-red scale. For all the visual benefits of the blue-white-red scale, it prevents the representation of missing data in white. I have seen this scale used either because the researchers (in the BioGeoChem-Model team) were actively trying to move away from the rainbow spectrum as harder to perceive, or (as in the RegionalNowcast-Model team) to denote that the numbers corresponded to a difference between two results being compared (so there is zero, positive and negative).

Smoothness in the image can be a meaningful signal: a "pixelated" appearance of a bounded color gradient suggests that this is a model result and the model uses this grid. Conversely, in a depth profile, smoothness suggests model results *or* measurements that were made in sufficiently similar conditions at short intervals and can therefore be represented in a continuous manner.

Transformations are ubiquitous and every transformation is subject to scrutiny via the charts it produces. Visual artifacts therefore have some properties and requirements to enable this scrutiny. They are:

- *Pervasive and flexible.* Visualization was the output of every act in the research pipeline. Though pervasive, it took on different roles in different contexts. Flexibility to be modified in order to be comparable with other data forms or sources was a crucial aspect.

- *Diagnostic and verbose.* A chart facilitates fast assessment of whether anything stands out, in the sense of looking "wrong" or "off." If it passes this diagnostic muster, the next step is to see whether it suggests anything interesting. Simultaneously, the chart is used to verify that individual expectations of behavior are met (e.g., that the surface

temperatures "look right"). The visualization must encode enough information to be an effective tool at different levels of interrogation. Sometimes, this means that its geographic or temporal range is limited.

- *Fast and reproducible.* Visualizations are embedded in a process of rapid prototyping and decision-making. For example, "which boundary conditions best reproduce observed patterns relevant for question or hypotheses?" Answering this question involves running the model with different boundary conditions, as well as running it for multiple timescales. In modeling, the decisions are about which parametrization, boundary condition, bathymetry, and so on, best reproduces the measures most relevant to the object of study. In observational data, the decision-making process aims to identify the best way to represent the data without mis-representing it. In both cases, rapid iteration allows to minimize spurious features in the visualization.

- *Intriguing and Exciting.* A single finished chart gets at least a half a minute of uninterrupted, silent consideration from its audience, be it in a presentation, or looking over the shoulder on the computer screen. It is intended to enable the audience to ask deeper questions that may require similarly-complex follow-up charts. In the case of observing over the shoulder, the follow-up charts may be created on the fly as a living embodiment of an ongoing discussion. As in Subsection 3.3.2, *The Beautiful Chart*, the visual products were the primary means by which a member of one team inspired a member of the other team to voluntarily set aside an entire day *that week* to do a lab task he would have otherwise not done at all, but in this case was quite excited about.

As an observer, I was struck that it seemed like the beginning and end of every discrete research endeavor was a chart: the initial provocation piquing interest; mid-task diagnostics, where a researcher might "convince myself that it works;" the final critique, a beautiful enhancement, or exciting support of a hunch. The chart deserving *wows* is the one which manages to use simplicity to communicate something complex about a phenomenon or pro-

cess is the ocean. The talks observed in course presentations, practice talks, and more formal completed presentations were each a collection of charts with commentary, often with a significant chunk of time spent using a wooden or laser pointer to identify regions of interest. In *Laboratory Life*, Latour and Woolgar claim that the iterative transformation of data into successive inscription devices is associated with increased abstraction [37]. In *Seeing Like a Rover*, Vertesi stresses the role of the image in scientific work as not only creating an artifact of a scientific process, but the scientific process itself.

Although many visualizations are only ever seen by one person, all visual artifacts incorporate elements that make them potentially viable in a social context. The ideal end result of a visualization is a complex chart demonstrating how multiple data sources come together to support a coherent causal claim. It is meant, therefore, to persuade a wide audience of inter-disciplinary researchers. But even the most ephemeral visual artifact is a persuasive tool, described as necessary for "convincing myself" that the analysis, or some part of it, works and the results are meaningful.

On many occasions, a visual object is produced by an individual to aid communication. The object in question has "weird" or "interesting" features, and the subject of discussion is resolving these features into a causal explanation. The result of the discussion is either an explanation of how the chart does, in fact, "make sense," or a concrete next step to investigate the "weird" feature or to further explore the "interesting" one. The level of scrutiny applied to both is similar, though the "interesting" receives enthusiasm where the "weird" receives a persistent, puzzle-solving sort of dedication. In this context, the key requirement is of the visual artifact to be easy to share. Examples given in Section 4.1 included arranging printed charts, or projecting in a group setting. The core design requirements here are ease, control and reconfigurability.

These interactions can be reminiscent of the aspect of *pair programming* which stresses a distinction between the person "driving" the computer, and the person who is engaging but purposely not physically manipulating the keyboard. Leaving the "driving" to only one person, in this setting typically the person who is explicitly asking for help from someone with

the same level of experience or more, accomplishes two things. First, it reduces the mystery of every step in the transformations or manipulations. The alternative is confusion and a sense of powerlessness. For example, one post-doctoral researcher notes that the "software advisor" in the group is very quick to fix the issue but seems annoyed and reluctant to explain, so that it is difficult to fix the same issue later. Driving, in certain contexts, may also reflect a recognition of an ownership of a problem-and-method combination by a particular person. While some questions from the colleagues (or supervisor) of the individual who made the charts can be at the level of guidance and explanation, many are questions to which the answer is only known by that person, who has spent time immersed in their idea.

This dynamic combines open-ended discussion and formal data-intensive analysis in a way that is difficult to support with "fancy" visualization software. I saw little "collaborative visualization" in the sense of multiple people working on a single visual artifact simultaneously. Mostly, the visual artifact was the product of a complex thought process of a person that was then used to reveal that thought process to their colleagues for the purpose of feedback. More importantly, the comparison and triangulation that is at the core of this exercise can be done either at the level of plotting additional data points for comparison or by articulating a known pattern verbally (e.g., "we expect to see this kind of salinity at the surface, but not in the deep"). This "informal" comparison that draws on shared knowledge - articulated as a hunch or as a whiteboard squiggle - is a crucial step of interpreting and interrogating a new visual data artifact. Remote collaborations that required visualization were sustained via static images and very long emails or video calls. These patterns of sharing were pervasive, creative (spanning a variety of visual elements for condensing and communicating data), and robust (not brittle once the artifacts are created).

4.4 Summary

In this chapter, I introduced the concepts of the working environment and the collective imagination of the perfect world. The *working environment* is articulated in terms of its technical components, but also includes cognitive and social components. In the case of oceanography

in particular, the highly specialized visual vocabulary shared within and across sub-field constitutes the cognitive component. The immediate focus of the working environment is on the visible aspects of code work: what programs are used, who is asked for help, when something is taking "too long." The collective imagination of *the perfect world*, on the other hand, refers to the open-ended, but directional, curiosity that drives events like workshops and skill-shares. Listening for phrases like "the ideal would be..." or "in the perfect world..." can reveal values, motivations, and priorities with respect to all three aspects of the working environment.

Visual display of data and results is a central component both of *what is* and the *imagination of what could be*. In Section 4.3, I review some social patterns involving visualization and some common elements of the visual vocabulary. The goal of these details is to provide context for the next chapter. In Chapter 5, I focus on that part of the perfect world that emphasizes the adoption of "best practices" from software engineering to improve the quality of products of code work. Throughout the section of that chapter, I include stories that justify *adaptation* of best practices (rather than direct adoption) because of some existing dynamic that is rooted in the role that visualizations play in embedding code work in scientific practice.

Chapter 5

BEST PRACTICES, BETTER PRACTICES

In the previous chapter, I introduced the *working environment* and the *perfect world*: the working environment is subject to change, which is measured against a collective imagination of what is possible. In the "perfect world," code is understandable, usable, and extensible (Fig. 4.1). In the "not ideal" present, it is some combination that creates a burden of "extra work" (see also Trainer et al [62]).

"Best practices" in software engineering span particular tools or social protocols that aim to ensure these qualities in the code that is produced. Some scholars working at the intersection of software engineering and scientific method development suggest ways in which software engineering best practices can be adapted to the scientific context (e.g., [70] is a particularly detailed list of such adaptations). Sletholt et al conducted a review of agile practices as reported in publication manuscripts, where the inclusion criterion included use of agile-related terminology. However, in their findings the authors commented both on practices that were identified as agile, and those which *appeared* agile [58]. Easterbrook and Johns find, based on ethnographic study of climate modeling software practices, that rather than lacking formal testing, the testing done is simply distinct from that in the software engineering sense, but no less 'formal' of a verification process [17].

Many of the "best practices" are more formal versions of practices that have arisen organically and which are not associated with the formal term. Pair programming is an example, as are other protocols for communication in programming practice, the subject of Section 5.3. I illustrate that it is possible to adopt a tool or practice without reaching the desired goals, illustrated previously in Figure 4.1. However, it is also possible to reach the goal without actually adopting a particular tool or practice. The execution of code over time

interacts with other factors that depend on scientific and social context. Human-readable artifacts degrade differently than those that are optimized for machines; though human-readability can sometimes have a very narrow audience that understands the artifact.

This chapter is organized into three sections, all of which aim to both describe an ideal and the deviations from that ideal in practice. The *ideal* includes philosophy, social practices, and specific tools in varying degrees, and each of the sections attempts to provide a description of these various components. Section 1 focuses on data as *a thing* that must be stored *somewhere*. Section 2 is about version control as a multi-faceted means for coping with time and the challenges of having multiple contributors. Section 3 covers a range of best practices that formalize communication in programming.

5.1 Where to Put What

In this section, I bring together a variety of concerns about data representation on disk(s) and in memory, which are coherent in that they appear in group discussions of *where* to put *what*, which concerns a variety of related concepts. **Meta-data** refers to the information that provides the necessary detail for reuse and is included in a standard by which data is made reusable in a shared **data format**. Zimmerman, for example, identifies other means, besides information embedded into data formats [75]. *Structured data,* particularly in database management systems (DBMSs) that abstract representation and programmatic handling from storage on disk (e.g., using a query language), are not particularly common in this setting, though they are used by the CustomInstrument-Lab. However, in addition to more philosophical issues of how data *ought* to be represented or stored are the more banal issues of *what to put where,* including things like parsing time-related strings, and grappling with small insidious errors as a result of misunderstood time-representation schemes.

This story begins at a meeting that had not yet started. It was an instance of a regular meeting between the members of the CustomInstrument-Lab. However, the collaborators, who were not working in this building and had to walk over, were running a few minutes late as usual, so everyone already here was casually hanging out and checking in with one another.

It had been a few weeks since the last meeting. The joint meetings of this group tended to have the quality, relative to other meetings within the members of the group, of a slow re-boot. People in attendance worked to recapture the momentum or shared conversation of the previous time. Mallory, a post-doc, came in, sat in a chair, and asked: "Who's coming? What's the agenda?" There was no urgent quality to the question, and Andrew, one of the people who played a Software Advisor role, noted in casual response: "We made some changes to names of functions," going on to explain that "No more files once you're done with [a necessary one-time processing step]." This generated an air of excitement, as George summarized the good news: "now we have a database!" and Mallory described it as a "conceptual shift." The shift, here, was not only in the *where* but also the *what*: a database is a layer of abstraction higher than a file-based system.

The representation of data can be encoded programmatically (data structures), through common standards (formats), and informally through inline comments, emails, or post-it notes-to-self. In this section, I first consider *formatting*, or how data is formatted from the vantage point of a user, such as a UNIX timestamp versus a human-readable time and date with a day of the week included. Then, I move on to *data structures* how data are stored and manipulated at a conceptual abstract level. These are related concepts because both are at core about how human-readable dates, measurements, and parameters are represented in the process of both tuning a model and running an analysis. In Section 5.3, I explicitly address debugging and testing practices, but for now, let us focus on that inevitable step of programming where an execution has resulted in something unexpected and the scientist is trying to identify the source of error by repeatedly interrogating what a particular value is at a particular moment of execution.

One way to do this kind of "sanity-checking" is through visualization of the values, a practice I introduced in Chapter 3 and to which I will return at the end of this chapter. This strategy, however, covers far from the complete range of errors. On many occasions, scientists who code have to go digging through code. At this point, the question "what is the value?" is answered not through specialized and dense visualizations, but through printouts

of data structures and through intermediate output (e.g., comma-separated value output). Even in the best of cases, this is a far less human-friendly format.

One of the most common sources of error, relevant to researchers of all levels across all four groups, is some error parsing timestamps formatted in a slightly unexpected way. This affects not only the data processing pipeline but also the semantic organization of data, as time-related information is embedded, in as condensed a form as possible, in filenames and folder structures. Additionally, variations in representing time include what to do with null/zero values (dense versus sparse representation). At one point, in the CustomInstrument-Lab, the error was that a student who was making a chart assumied that the database had no missing rows of this sort and therefore could be aggregated three adjacent rows at a time to make their big data more manageable given the specific research question. However, there were gaps where collection had been stopped for technical reasons: so her assumption, having not gone on the cruise and been as familiar with the data as those who did, is understandable. This resulted in obscuring a very interesting finding that was sensitive to daily patterns.

Consider also the following exchange in the CustomInstrument-Lab team, between a software advisor, Andrew, and a post-doctoral research fellow, George. Andrew is explaining a recent fix to George: "oh, I made it so file names are consistent - new function, [foo], which detects old vs new style and formats folder and file itself." George clarifies: "so user doesn't have to do this?" And Andrew confirms, which seems to make George quite happy. As they move on, George walks Andrew through a function adopting the familiar second-person voice ("and here you do [this] to [that]") and Andrew looks over his shoulder. At a particular point, Andrew points out, incidentally, since the topic had shifted somewhat, a place where a particular function is relevant. George laughs and asks: "why is this happening *now*? Didn't we fix it?" Andrew explains, with George recasting his understanding in less technical terms to confirm that they are on the same page. Andrew sighs: "'I wish we never changed file naming convention..." George does not let this remark slide unnoticed: "oh no no no!" but Andrew reassures: "no, I mean, I wish we *started* with time-stamp." George, reassured successfully, tells a quick side story about another researcher who insists on using

an unreadable numerical time-stamp. Andrew agrees with the implicit verdict: "it's good for computer, not good to access as a person," going on to explain that "most libraries interpret correctly despite multiple standards for date formats," so there is really no advantage to the numerical representation.

"Technical debt" is another term, like debugging, which was applied virtually not at all by the scientists to themselves, but which appears routinely in discussion of programming practices by computing researchers and practitioners [35]. Data formatting decisions constitute a visible form of technical debt: burdened by the inescapable wisdom of hindsight, the oceanographers observed expressed the most regret with regard to how data were stored and accessed. The frustration of time sunk into shuffling bits around inefficiently is very different from the experience of analyses scripts that may seem inefficient from the outside. If some thing takes an hour or two to run but could be re-written to be only 15 minutes, it might be not problematic at all in some cases: if the procedure is robust enough to execute without supervision, allowing the researcher to launch a process and eat lunch, go home, take a break, or transition to a different tasks. If there is a more efficient, faster version, it is only better if it also requires no supervision. If the faster script is also occasionally buggy, or somehow less certain in its correctness, it demands more researcher attention and time than the computationally-slower alternative. Inefficient data parsing, writing, and processing, however, always demands attention because it is typically a work-around an upstream problem: the original data format is subject to the peculiarities of the process and team that produced it. Even with standard accepted formats, like NetCDF [53], some interpretation is necessary.

I was interviewing Melissa, a computer science student who at one point had been part of the CustomInstrument-Lab collaboration. Her involvement was winding down by the time I began my observations, so in interviews I mostly asked reflective questions. After one such interview, we continued the conversation and she tells me how she sent Argo data to

some UW researchers working with Argo floats[1]. Articulating her troubles finding a group of physical oceanographers to collaborate with, she told me: "I just hoped they would be more excited, but they were getting caught up on things that I didn't think were important." In particular, she had sent them something she had made based on the data, and received in response an explanation for how a particular column ought not be used for data (it is only used for quality control). The column names had been named without enough verbosity in the NetCDF, so the common format did not save a capable and enthusiastic researcher from violating unstated assumptions. This was frustrating for her, because her intention to demonstrate a visual artifact process had backfired in a way that she felt was disproportionate to the difficulty of fixing the issue: "this is a *database*, we can filter that out later! I was *just* plotting something!"

During a shadowing session with the RegionalNowcast-Model team, Erin spends some time looking at data published on the website of a particular service which makes available daily weather data, relevant for comparison against their short-term forecasts (nowcasts). Her goal is to "figure out how to parse it into a format our models like to read..." She continues: "but before investing time in that, I would like to figure out if the data is good enough. This is a task I haven't been looking forward to doing - it's just not fun to parse data from files. The first step was to paste data from file into my [iPython] notebook, that was a waste of time, I've given up on that. [The idea was to see if I could] just look at it quickly without writing a parser? Not the way to go, can only look at one day, and the location I chose is not close to what I care about. So it was a big waste of time [to attempt to avoid writing a parser by processing data 'manually']. Now, I will write [a script] to parse [the needed third-party data]."

Provenance information is indispensable, but it may not be clear which parts are actually necessary to record until it becomes necessary. The technical capabilities of the data storage format can be perfect but if the limitation is conceptual, will still bottleneck on emails to

[1] "Part of the integrated global observation strategy," a project that now offers 15 years of global float data. See http://www.argo.ucsd.edu/

someone who was on the cruise. That said, the technical implementation is often far-from-perfect. The format that is "simple" and "readable" to produce can also be the one that generates more work for parsing. While talking to a junior (undergraduate-level) research assistant who was working on "cleaning up" some of the code, commenting that "the hardest thing is to make [the output] user friendly" and "access to type of input someone could use." I ask if there are any standard formats, and at first she is confused but then she laughs: "no. That would be nice, but not that I know of."

On another occasion, Omics-Lab group is discussing in a meeting an especially interesting and relevant new paper with a cornucopia of data and figures in the appendix. However, the accessibility of something does not necessarily mean knowing enough about data provenance to use it. After spending a bit over an hour collectively discussing and interrogating the paper, its figures, and even opening the attached (partial) data in a spreadsheet, the conclusion is to contact some of the people in the team that published the paper to learn more how to work with this data. Ad-hoc, immediately readable formats are common for the same reason that inconsistent time-stamps are common. Using standard formats introduces a lot of structure, which is at best unnecessarily verbose and at worst brittle and easy to render totally useless with errors. Intermediate output is vital to a particular form of debugging, but may be the among the first things rendered inaccessible by overly-oblique formats.

So far, I have provided examples of issues that come up relating to *how values look to a programmer or user of analysis software*, not so much *how values are represented computationally*. In a programming environment where the "what is the variable's value now?" question is primarily answered by printing out the variable to shell or to a file which can be monitored as the operation progresses, the format and the structure are more connected than with database management systems that provide more abstraction or even with complex meta-data rich structures (extracted from standard files) which are impractical to print in full during debugging.

The separation and abstraction of a DBMS allows a certain degree of power but also shifts the locus of control and scrutiny. If it is beyond the scientist's programming skills,

it is qualitatively different that being merely less efficient or less powerful. When the CustomInstrument-Lab experienced data loss due to a technical issue with the software and hardware data infrastructure maintained by their computer science collaborators, the data was not "lost" in the sense of "gone for ever", the way that a water sample damaged in shipment might be lost, but it was rendered inaccessible enough that the scientist most affected by the end said she had spent an additional month trying to get the data back into the database. The person who left in the story "The Departure" in Chapter 3 was the person who had gotten the data in there. Because he was not available, that work had to not only be carried out again, but carried out with a sense of failure and frustration.

The complexity and size of the database in the CustomInstrument-Lab group arose from the complexity and size of the the dataset itself. A post doctoral fellow on the project, in an early interview[2], identified "the package that read raw [instrument] data, binary data, read them into something that makes sense" as one of four R packages he uses in the course of the work. Indeed, refactoring this particular package was the first joint project between the oceanographers and the eScience researchers on campus. During my first observation with the team, I was given an introduction of the code, which has been "built in-house" for two major analysis tasks: analysis during collection (on the cruise), and analysis afterwards (comparisons and longer-term storage). A senior researcher explained to me that they wrote some code 3 years ago, at which point others, who were also present in a small joint meeting regarding this software in particular, chime in with jokes about how "crappy" that code was because it "didn't involve any computer scientists" and how now the computer scientists collaborators are helping to "deal with a lot of bad programming."

Other than the CustomInstrument-Lab, no other teams used a DBMS. At some point, as a particular meeting of the Omics-Lab wore on, it dawned on me that the participants in this particular meeting were using the word "database" to refer to a file. This was jarring to me: the term "database" overwhelmingly implied that the storage and manipulation

[2]One of the ones from Spring 2014, conducted by Charlotte P. Lee's group, prior to the beginning of my study.

of bits on disk is abstracted away from operations over the data. In this case, though, the textfiles contained sequences of interest and were an embodiment of careful researcher-specific curation. Colleen notes that [colleague 1] is using [colleague 2]'s database, not GAWS, which surprises both the lab manager and the PI. Colleen is, in turn, surprised by their surprise: "so should I use this dataset curated by him?" to which the PI suggests: "I think you should talk to [colleague].... this is like her lifes work, to make these databases, and she's right there."

Study participants spent time and energy discussing issues of making their own data available for later and/or broader re-use. The data access endpoint can offer the following features, each of which can be noted above:

- Online access, with any necessary permission restrictions automated. Removes the need for an email exchange to facilitate data access.

- Programmatic access, through enabling URL-hacking or an API. Removes the need for manual download of data.

- Integration of multiple data sources and relational joins to allow users to download only that data which is necessary. Helps to alleviate problems of big-ness; the whole dataset may be unmanageable for a user, but perhaps the data actually needed is much smaller. Removes the user inheriting some of the "big data" problems. In practice, this can be achieved using a form with dropdowns or other UI elements, which may or may not co-exist with programmatic access in the above point.

- Surfacing of summary information about the data, including in visualizations. Removes the need for shot-in-the-dark trial-and-error (and/or emailing someone) in the above case.

- Timely access, relative to when data is collected: real-time from cruise, immediately after cruise, every 6 months when data is collected on a USB-drive from a partic-

ular location. The complexity of annotation and ingestion of data may improve its accessibility and usability, but it may reduce its timeliness.

All these features are orthogonal, and the range of their various combinations are represented in the data endpoints for oceanographic data. In addition to this plurality of mediated data access sites, increasingly common programming practices mediate the way in which these data are produced or manipulated. The examples in this section revolve around issues of *where* and *how* data is stored for computation. Ad-hoc, unstandardized formats arise from an immediate need for trouble-shooting: short, readable time-stamps at the right granularity are helpful for debugging, but not as helpful for moving forward. Because file-names and folder structures are used extensively for data management, and because *time* is a key component both for analysis and the reality of data collection (in addition to the component of geographical *region*), decisions about time stamp readability affect decisions about file-management and therefore data-management. Whether or not the word "refactoring" was used to describe the act of changing time-stamp or file naming convention[3], it was as unpleasant as it was necessary. It is not clear if using a database solves the conceptual problem here. A DBMS instance is far more brittle than a collection of files, which is relatively more understandable with simpler tools. The group that used a DBMS did it to dramatically extend their capabilities to work with more data. For analyses that did not require this resource, the scientists would occasionally do analysis on their local machines having downloaded the right view.

5.2 Version Control

Version control is by far the most universal best practice that everyone feels bad about. If we define the best practice as an acknowledgement of something they "should" be doing differently, then we do not need to spend much time observing scientists talk about code

[3]common and crucial for organization of data

before concluding that *git* and *github*, and less common but also acceptable *hg*[4] and *bitbucket*, are best practices whereas no version control or *svn*[5] warrant explanations and apologies. The PI of the RegionalNowcast-Model group - the same one who keeps the team's software advisor "in check regarding the complexity of his solutions" - comments on her preference for hg over git, to the point where a departmental workshop on version control which she organized has adapted SWC materials to hg: "well to be honest, just because I'm more used to it, but also hg is a more limited, simpler, accessible thing for people starting out, but I would never tell anyone to switch." So I ask if she would ever intervene if someone was using svn, which she considers for a few moments with hesitation: "*weeeellllll*... I would suggest that hginit is really small and lightweight[6] and there is nothing lightweight about subversion."

Everyone is controlling versions through social agreement, such as using email, files, and access to a common machine. However, when a scientist is asked whether they use control and answers in apparently-apologetic statement of not using it, they refer more specifically to a utility designed for managing work on complex, evolving codebases over time (usually, git). The question and statement can also refer to using a particular service, like GitHub, which offers hosting for project (though public), and which also has a desktop client with a GUI to make using git more user-friendly than typing commands in shell. The "organic" version control, the kind that is not referred to as "version control," consists of carefully (or sometimes not so carefully) organized folders with a naming scheme peculiar to the small group of people affected by them. In practice, adopting software version control can be a case of tooling solving a different problem than the one people are having. Tooling is not sufficient because tooling is designed for use with code. For GitHub, for example, it is essentially equivalent to other directory-based methods of version control, and less

---

[4]pronounced: mercurial

[5]pronounced: subversion

[6]here, she is comparing the relative complexity of initializing a repository - the "getting-started" act

straightforward when it comes to large data files or images. In the case of those things, the code may actually be the lest complex and noteworthy component by comparison.

Furthermore, what is "technically possible" is sometimes not what is actually understood and done. According to a conversation between oceanographers and some searching on forums.ni.com, there is a version history tool in LabVIEW which can be connected to a VC system but someone saying that is far more likely to also say "but I don't know how it works really" than to actually give directions. The issues particular to LabVIEW are beyond the scope of this work, but a researcher asking the question of "but why aren't people using this excellent technical capability that they recognize as present and useful?" can, I hope, find some of the conceptual frameworks in this dissertation applicable.

The question of *what version control actually is* is not a trivial one. As with other things in this chapter, there is a distinction between the *thing* - the specific set of modifications to the working environment necessitated by adoption - and the *reason* - the desirable general outcome. Version control is of uniquely universal interest to this setting, relative to other programming practices and concepts covered in the next section: everyone in the study was either using version control, feeling bad about not using version control, or had a thought-out position on why using version control is not for them. For example, having a minimal amount of code, with all the complexity embedded in the data or the mathematics, suggests that though programming is important to that work, the investment of energy into learning to use a particular version control system would not be worth the resulting unnecessary work-flow complication.

In conversations about version control, novices may conflate different concepts that have all become entwined with the persuasive language around version control in SWC and SWC-like contexts. They may use any related terms to invoke the whole idea of version control, and likewise interpret "version control" to mean any of those things. These terms ranged from specific tools or processes ("merging," "branching," "git," "GitHub") to broader *team or field* level terms ("best practices," "standards in my new lab," "collaboration with other institutions," "online professional portfolio," "notifications and passive awareness of other

research projects") to larger philosophical principles ("accountability and transparency to stakeholders," "open source software"). The collapse of these disparate terms into one unified, if ambiguous, notion of version control was revealed when participants - especially those I recruited via SWC events - began talking about something very specific and ended up talking about something much broader: three people, out of ten, said they did not want to use versioning (a specific process) because they did not want their code to be public (broad philosophy), despite the latter not actually being a necessary outcome of the former. All this supports the idea of version control (and related topics) as a kind of gateway best practice, topping an interminable to-do list of ways in which software practice and discourse influence scientific code work.

As mentioned in the prior section, the organization of Software Carpentry curriculum intentionally folds broader philosophical ideas (collaboration, accountability, reproducibility) into a discussion of concrete tools (GitHub). This is because it is easy to get scientists to be excited to learn about tools, but they are not as open to going to a workshop about how to do science, even though the tools and the practices are interdependent [69].

A post-doc in the BioGeoChem-Model group says she is "trying to use github from an earlier stage," which is new for her; she only recently started using github, and so far it has been to make public her more established work. "It's weird," she says. "I never used version control, saw people use svn, and I didn't see that as useful." So she embedded versions in file-names: "everyone says it's a bad way to do it, but it worked really well for me. I never lost anything." She saved new files at "checkpoints," which she says are like commits except that commits are more frequent than her saves. GitHub seemed better than svn (subversion), so she had "less of an excuse" not to do version control.

When I first ask him about it, the Software Advisor of the BioGeoChem-Model group tells me: there is no version control. It is shared by being on the computer to which both the people who work on it directly have have access, which is sufficient for their needs. Her colleague, also in the BioGeoChem-Model group, comments that while she uses it, he does

not: "I did use git with bitbucket[7] until I tried [uploading a too-large file, got locked out]. Tried using it again, was locked out after putting big data files, tried to fix with tech support but lost everything." He says he did not use version control in grad school, but "had an organizational problem that was nothing version control would have fixed." At grad school, they had a good back-up solution that was "good enough" but now in its absence he is using version control because it is free. In grad school, he tells me, back up was a big deal: "there were horror stories: 'sprinklers go on in the lab, so have to photocopy/scan the lab notebook!' "

At some point while I was shadowing a post-doc in the RegionalNowcast-team, she accidentally erased the data she had been working with on the previous day. "The data was sitting in the [iPyNB] cell, in this mess of code, I just deleted it." Mildly displeased, she went to previous commit, from last night - before she left, she committed and pushed. She goes on to tell me: "I find everyone who comes onto this project has never used version control before - I hadn't - but it makes everyone's life easier." Indeed even when I had first begun this study, she was far less comfortable with version control. Besides version control enforced for all, there is considerable discussion and standardization of file structure, and *where* things should go in this team - to avoid the kinds of problems that version control cannot fix.

A wide range of version control practices are used to cope with the organizational challenge of managing a variety of precious data and its associated processing or analysis code. There are privileged forms of version control which appear mostly as unattainable "perfect worlds" that either do not offer enough benefits or do not present with a clear course of action to inspire adoption. Nevertheless, these form a major part of the shared imagination, awareness, and intention.

---

[7]another cloud based service for repository hosting, like GitHub. Supports both hg and git, and has the added benefit of free unlimited private repositories for academics, whereas GitHub forces public repositories past a certain number of private ones. This begins to really matter if a researcher organizes her code with one-repository-per-project, as there can be a large number of ongoing or hibernating projects.

5.3 Communication in Programming

In this section, I address several different subjects spanning various practices in programming that emphasize - or consist entirely of - social practices and communication protocols. These are similar to the software engineering concepts from the prior sections: they are (overwhelmingly, though not categorically) not the terms that are used by the participants, unless they are explicitly incorporating what they see as useful software engineering lingo. An outside observer perspective can interpret both the presence and the absence of these practices, depending on how strict the interpretation and how much formality is desired. These concepts are arranged in a loosely-chronological order relative to a piece of code being created. These particular practices were included because all were present for at least two groups in the *informal* variant. Their more formal (of varying degrees of increased formality) variant was at least the subject of discussion - if not implementation - for at least one group. Unlike issues in the prior two sections, these "software engineering interventions" are even farther away in terms of obvious need: it is unclear what the costs and benefits are, and whether the solution is well-suited to the problem at hand.

There is a pattern of development where there is a "working environment" corresponding to *exploration* and one corresponding to *replication* of now-reliable (or reliable-enough) code. In the former, the scientist looks at a split-screen of script and execution environment (in MATLAB). In the case of exploratory analysis, such as in the iPythonNotebook environment, the trouble is that at the point where a test case can be articulated, half the battle has been won. There is a lot that still has to happen before that. A lot of testing is basically "sanity checking" which is necessary because everything is so brittle and unstandardized that there is no sense of trust or faith in the code. This is a persistent, daily paying of the "technical debt."

5.3.1 Design, Maintenance and Refactoring

In listing best practices for scientific computing, Wilson et al suggest to "document design, not mechanics" [70]. However, in their survey of claims about software engineering practices in scientific development, Heaton and Carver (2015) point out that design is not seen as a distinct activity in scientific development, especially modeling. In other words, the code disappears relative to the underlying math and scientific exploration. This increases the difficulty of documenting the design of code as a distinct (and distinctly error-prone) layer. More formal design would involve making charts and illustrations of the software, not just of the math (ie, equations) or the hypotheses (eg, whiteboard squiggles of depth profiles).

5.3.2 Documentation

When it comes to contextualized persistence over time and incorporating multiple research products into a coherent statement, email wins out over inline comments and over API-style documentation[8], so it is no surprise that it is a much more pervasive form of documentation. Modular and inline documentation requires that the users be able to not only call up this documentation in their working environment, but also have recognized that this particular documentation is what is necessary. Although modular documentation supports larger codebases with more contributors, it demands continual discussion of its relationship to other forms of documentation and non-code artifacts in the research context. Furthermore, an email remains "frozen in time" in a way that a collaborative codebase does not. As a result, documentation, if it exists in code, assumes an ephemeral and brittle quality. Whereas the practice of searching an email inbox is now near-universal, the same is not currently true of version-controlled histories.

---

[8]for example, doctrings in Python. Correctly formatted, these come up as tooltips when you type names of your own functions later.

5.3.3 Debugging

In Chapter 2, reviewing literature oin the section about software engineering in sciences, much literature stresses a lack of design and debugging as distinct steps in programming and the collapse of the code and the thing it represents (the math) into one entity that sometimes works and sometimes does not. A few of the participants used a debugger, mainly those with a computer science background. An oceanographer who was actively trying to incorporate a debugger into her FORTRAN environment ultimately was not able to find a better solution than print-statement debugging, which was by far the most common type (aside from debugging-by-making-charts) and is addressed in the prior data-formats-and-structures section.

Debugging of some form is unavoidable, and there is not a well agreed upon single "best practice" of a particular style of formal debugging that is widely enough adopted. I have included it for two reasons. First, an overarching claim in this section is the idea that each of the programming practices listed is inevitable in even the most ad-hoc programming practice. Second, all these practices are varied, some more than most. Ultimately, I include it in this list because of the claim developed in the prior paragraphs on Documentation and Design: in the scientific context, it is possible (and common) to do code work without recognizing code as a distinct element from the science. This results in particularly informal[9] debugging (and testing) practices. And as I mentioned before, "debugging" is not the term that participants used themselves.

Formal methods of debugging include: (1) using debuggers in the development environment for runtime debugging; (2) navigating between declaration and usage of a variable or function in the development environment, in addition to the step-through in a debugging-mode execution; and (3) pre-emptively anticipating possible violations of assumptions with descriptive exceptions and errors in the code to communicate the state at which a failure

[9]In this section I use the terms "formal" and "informal" relative to the software engineering formalisms. Many of the "informal" practices, such as the visualization-based debugging and testing, can be seen as valid alternative formalisms [17].

occurs when it does, rather than iteratively adding print statements to backtrack the error after it occurs. In the particular groups studied, these practices were adopted minimally if at all, paling in comparison by frequency of use to rapid visual examination of a graphical/chart output (which is debugging the science and code combined, not the code by itself).

The first two debugging methods require a more customized and complex development environment than any of those I saw. In the cases these practices were used, the scientists specifically said that it was fortunate that they were granted the necessary time to get "set up" with a good working environment (a few days to a week). In all these cases, a trusted Software Advisor helped to set up the right environment. Even the most seemingly-basic aspects of an effective programming environment, such as syntax highlighting, require active modification of an unfamiliar tool (e.g., shell or emacs). The difficulty of this is palpable in the first half day of each of the SWC-like workshops observed.

The last method - using exceptions and errors to allow communication, either between the program and its user, or between the programmer and her future confused self - is challenging specifically because it requires approaching code on its own terms, not the terms more familiar to the researcher (i.e., the math or science).

5.3.4 Verification and Validation

Although software is rarely recognized as separable from the mathematical or analytical concepts it embodies, it is still subjected to fierce, continual scrutiny (which is also reported by Easterbrook and Johns in a study of climate modelers [17]). Heaton and Carver draw analogy between software engineers and code to scientific programmers and math in order to bridge the terminology of validation: "Software engineers typically understand Validation as the process of ensuring that the project specification (and overall end product) matches the project goals or user requirements. In scientific software, this same concept is described as ensuring that the mathematical model (the equivalent of the project specification) matches the real world (the equivalent of the project goals or user requirements)" [27].

5.3.5 Testing

Formally, testing refers to a suite of declarations of what output a function should produce when given a certain input. Tests are used to check that code that is undergoing changes or additions remains correct. In the Agile programming paradigm, for example, the tests are supposed to be written before even the code, as this paves way for concrete and measurable goals. Many frameworks exist to automate tests, with manual testing sometimes necessary but ideally avoided to the extent possible. Software engineers distinguish different kinds of tests, such as unit tests (does this function, given the correct range of input, produces the desired output?), integration tests (do the different components of the complex program fit together?), and validation tests (does the entire complex artifact perform well relative to the initial specifications?). Informally, "testing" is that series of actions that "convinces" the scientist that her code works and that she can move on to using it in answering a scientific question.

Generally, the "informal" variant of testing involves generating plots and charts that demonstrate that "it works." However, if we focus on the lack of software engineering distinctions (unit versus integration) and the "manual" nature of looking at charts *we easily overlook* that such testing can (1) embody rigorous distinctions (boundary conditions, comparisons to different sources of data versus assumptions and self-checked math) and (2) expertly combine the human cognitive capacity to quickly interpret well-designed graphics with the need to operate in an area of great uncertainty, where things that are unexpected are either errors or discoveries, and if you knew on the outset how to tell them apart, you would have done this already.

One of the four groups studied began to write formal tests toward the end of the study. Their adoption of testing is described in Chapter 7 as one of the illustrative case studies.

5.3.6 Pair Programming and Code Review

The term "pair programming" was occasionally noted but not really owned by anyone, though I witnessed almost everyone engage in the hallmark behaviors of pair programming. One programmer, the "driver," sits at the machine while the other sits without touching the keyboard but asking a lot of questions. This process forces both programmers to articulate their assumptions and implicit knowledge, and this type of pair exchange was very common both among peers and between the more junior (driving) and the more senior (question-asking) researchers. This is distinct from, and complementary to, the dynamic where the more junior researcher watches as the more senior demonstrates how to do something. When I was giving the exiting-the-field talks, where I had summarized some of the intermediate forms of my ideas on formal versus informal software engineering practices, and I pointed out in a brief aside that many of the interactions I witnessed can be seen as pair programming, one of the (more senior) oceanographers responded by rejecting the term, not combative of me but in a move similar to "I am not a real programmer." Not a declaration of wanting to be a programmer, but the opposite, a distancing.

Consider the following situation, where a junior student was working on an analysis script. The script was not run in full; parts of it were highlighted and executed, simulating a modularity that is not strictly part of the script design. She was using MATLAB. Each section executed modified values used by another section. The execution record became convoluted and when a resulting chart showed an unexpected peak or trend, the student was stumped. This was not her code, she was nervous to have me there, and by way of explanation she muttered "I don't know what [another, more senior student] was doing here." Her trouble understanding the code was superimposed on her trouble understanding the intention of the original writer of the code. Eventually she was stumped enough to fetch a senior member of the lab, George. He came to help, sitting behind her and asking her questions like "what does [this variable] do? where is it set? where does it change?" Repeatedly, she generated charts and he asked her to interpret them, to spot the errors.

They would not call this "pair programming," because there are other components to this interaction: the more senior researcher is mentoring the more junior one in a way of thinking. This dynamic includes training by example and verification using visual products.

More formal code review refers to systematic walk-through and feedback. This might be in a pair programming context, or through something like GitHub's annotation and commenting feature on commits that are submitted but not yet integrated into the overall codebase (pull requests). Formal code review is a mechanism that allows a contributor to the code only modify or add to the codebase with review from an individual appointed either by seniority (of skill), expertise (in the particular element of the codebase), or ideally both. Using GitHub, a contributor external to a public project may fork that project, and then request that the changes she made be merged into the codebase. This request triggers someone internal to the project to review the code, potentially making comments and requests for modification before approval may be granted. This is a very particular implementation of peer review. Although not formal, Github-style code reviews took place during my study, people solicited and provided feedback on code, as well as associated a sense of authorship to pieces of code and their functionality. Rather than frowning, "this makes no sense," the scientist can just as well quip, "I have no idea what Alice is doing here, I better ask her."

The contexts I studied were intertwined with teaching dynamics. "Pair programming," to the extent that it took place during my observations, was a means for expertise exchange. As in many other contexts, the experts included a deliberate effort to help the more junior researcher or research assistant develop a sense for how to spot things that are strange, a taste for things that are interesting, and an intuition for what questions to ask to visually interrogate their data or model using a series of graphics.

The RegionalNowcast-Model group has a lot of practices in place, some of which include software engineering best practices, but many of which are organizational best practices that allow most efficient use of the scarce resources of the PI and their part-time technology advisor Ed. In Section 7.2, I include a story about a one-day "sprint" that the group hosted

for themselves, and the annual SWC events they hosted for their department. Communication around code included other teaching and scientific-feedback patterns, such as "show and tell." This dynamic was apparent in many meetings (some of which were actually called "show and tell" meetings, hence my use of the label to describe a common occurrence more broadly.)

5.4 Summary

Claims of how software engineering practice can beneficially inform how scientists do code work seem to assume major, or even fundamental, differences between "typical" software engineering and scientific coding. In Section 2.1, I include a review of ways in which existing literature has delineated scientific work as separate from software engineering in more typical contexts (e.g., business). Code work in science, aa it has been studied [17, 32, 45], is specialized and varies by context. However, so does "typical" software engineering, which contains "amethodical" practices despite a rhetorical emphasis on the "methodical" [64]. The emphasis on "best practices," which are in constant flux [58] and open to a degree of interpretation can be seen as "moral claims of practical necessity" [71].

Section 5.1 on formats and data structures united those terms over challenges in how a scientist determines the value of something during trouble-shooting an unexpected behavior. Section 5.2 on version control considered both version control of software, of data, and of projects, as well as the extent to which these use cases are covered by the standard pitch for GitHub. Section 5.3 covers a range of different subjects, pulled together by the common thread of *communication and feedback* in code work, spanning design and maintenance.

Getting data representation right on the first try, in terms of data structures and formats, is an impractical, if not impossible, goal to tackle. Non-ideal decisions in this area accumulate into problems over time, as researchers attempt to improve the data representation schemes and encounter compatibility issues with prior runs. In a perfect world, version control would help ease the pain of progress and change. In the practical reality, though, "version control" is a behemoth philosophy with all manner of baggage. It bears with it not only the bash

shell, but the cloud, open-source, open-science, and even professional identity management. Regardless which particular version control tool is most approachable, the entire mass of intertwined demands can make it seem like only a massive overhaul of work practice will do, as if no incremental improvements are possible without commitment of time and resources that are simply not available.

The other programing practices - those covered in Section 5.3 - all share a quality of being less immediately-relevant than version control, as well as being (like version control, in fact) various levels of formalizations of practices that already routinely take place and arise out of the normal skepticism, feedback, and teaching that are omnipresent in scientific work. However, formalizing them by deliberately and explicitly incorporating software engineering narratives and aesthetics can be a hard sell: cost and benefit are hard to determine ahead of time. Moreover, teaching and mentorship is a major part of what the PIs and research staff do, so the costs include not only the cost of that researcher embarking on an unknown path, but also the cost of then guiding a hypothetical future student along that path.

It is possible, from the perspective of a software engineer, to look at a broad range of code work in science, be understandably perplexed, and believe that this would all be far easier if only the scientists who code would adopt some or all these tools and processes. The main thing that everyone agrees on regarding the SWC curriculum - students and instructors alike - is its ambitious and dizzying breadth, and the difficulty (impossibility, according to some) of covering it all in the allotted time. But even in designing their own SWC-based curriculum, the oceanographers of the RegionalNowcast-Model team were unable to condense anything, or take anything out. The only thing they left out were the databases: they did not use them. But all the tools that they *had* successfully incorporated into their work were now indispensable and inextricably connected.

The opportunities for incremental improvement exist in the very area that receives relatively little attention compared to version control and data formats: the formalized social protocols in Section 5.3. Furthermore the teams most successful in incorporating version control and particularly effective data organisations described their solutions as “just” whatever-

they-were: nothing was the golden ticket, the silver bullet, or the stone that would get two birds. They managed to avoid the overloading of one concept (e.g., version control) into many distracting ones, and this helped make steady progress in manageable steps with minimal time-consuming overhauls. RegionalNowcast-Model group had adopted an impressive number of the best practices that they value. During the course of the 18-month study, Lindsay, a post-doctoral researcher, went from beginning version control to being motivated to explore automated testing frameworks. They did not start with best practices, but instead built up different skills piecewise and over time.

Everything is project-centered, and many things start as "interesting explorations" or "side projects" which are not deemed to be in the purview of whatever advanced organization solution other more "mature" projects are in. As a result there is a moment when a side project graduates to being an important project where old "technical debts" must be paid. There are several way in which the best practices addressed in this chapter outlined above is insufficient in scenarios where (1) visualizations are pivotal conceptual/organizational elements and (2) reproducibility and automation can create a tension around what does or does not need to be abstracted out and encapsulated.

The three sections in this chapter offer a view on various "ideals" and deviations from these "ideals." The selective, sometimes deliberate deviation from these ideals on the part of the programming scientist who is "not a real programmer" is indicative not so much of a failing but of a different concept of effective persistence - and what effective persistence *requires* - in a specialized context. In practice, the conversation of what scientists ought or ought not do in the realm of code admits that the context is different (e.g., [27] [27, 54, 12], but I add a more concrete way of thinking about these differences that validates the informed choice to adopt, *adapt*, or altogether reject a recommended "best practice."

In some situations, there are problems that *could* be addressed by best practices, as well as a recognition of this, but without a sense of how to proceed *incrementally* without a wholesale overhaul of existing practice. Version control, especially GitHub adoption, is by far the most common of the particular best practices to be mentioned (often in the "I

should be using _, but I am not" context). However, the most effective practices as they were adopted by participants were formalizations of the communication patterns described in Section 3

What is the test for whether a particular "best practice" has been adopted, or adopted poorly, or simply evolved naturally without any intentional adoption of it at all? Is it important for something to be intentionally adopted for outside observers to then measure how well it is working as an intervention, or does it no longer count as an intervention? In the next chapter, I delve into a discussion of *deliberate* acts of change.

Chapter 6

MOMENTS OF FLUX

Previously, Chapter 5 paints a picture of change as incremental, with adoption of "best practices" arguably reducible to a question of perspective or interpretation of existing practices. This telling uses the concepts introduced in Chapter 4 of the *the working environment* as subject to small acts of deliberate change, informed by *the collective vision of the perfect world*. The arguments in these prior chapters included defining "better practices" as ways in which existing environments are intentionally subjected to iterative change with the perfect-world goal of incorporating best practices into contexts that would benefit from them. In many cases deliberate change and adoption are more involved than including new vocabulary, though expanding the language of expertise is part of it. In the cases of more substantial alteration of the working environment in pursuit of the perfect world, the "leap" involves the risk of uncertainty, where the costs and benefits are difficult to assess. This chapter addresses the antecedents and mechanics of this leap.

Throughout this chapter, I examine the act of deliberate change, which I refer to as the *moment of flux*. The focus of this analysis is decision-making that is demanded by ubiquitous opportunities or challenges that arise continuously in code work. Each day for each scientist contains many multiple moments of flux. Through stories of adoption and adaptation, I will examine deliberate acts of changing the working environment, in the context of practical constraints as well as idealistic visions of possible futures. Consider the following story, from one of the interviews with coding scientists who attend workshops for programming. This participant attended several different types of intensive educational initiatives. Although he did not attend SWC, and he read about my study on SWC-related twitter and email

distribution lists. In his interview he reflected on various other professional meetings aimed to introduce programming tools and skills to scientific code work.

As he first described the "very informal, one-hour lunchtime meeting of users of [a particular] system," we see the role of the community for building *awareness* of possibilities: "Usually there is a tech talk, which is about half an hour and the other half hour is just people talking about their experiences [with the system]. If they have any tips or tricks that they want to share with the group, they can do so. Also, some of the administrators and the high performance computing experts .. are present at the meeting [so] you can have a quick consultation with them. I usually attend the meetings because it's an opportunity for me to do a little bit of reflection on how I use the cluster. Also, there's a lot of really interesting information that ... can be gleaned or tips you can pick up from chatting with other users and seeing them present their research and specifically how they accomplish their research on a shared computer cluster."

He described a particular time when his *intention* (for finding better data storage formats, as evidenced by trying different things) turns into *action*: "Prior to [a talk on how to use HDF5[1] with the HPC system], I used a lot of different ways to store my data. I used MySQL databases. I have used FAT files. This talk about how and why HDF5 is a better, safer format for my data really convinced me to **take the plunge**. I'm pretty proud of it, and it has saved me some headaches certainly since that time." As I continue to ask him about "taking the plunge," he tells me: "It actually wasn't quite so extensive as [that phrase implies]. It was really just the process of changing a few scripts in terms of how I load the database... It was quick, it was **quite painless**. I didn't have to do a whole lot of work. They were already Python packages, for instance, like PyTables and H5Py for interacting with HDF5. It **wasn't nearly so bad as that**."

Despite this participant's insistence that "the plunge" was not "nearly so bad" and indeed "painless," the language he uses to describe this is characteristic of the way in which certain

[1]HDF5 is a data format that is well-suited for storage and manipulation of large numerical datasets from NumPy. See http://www.h5py.org/

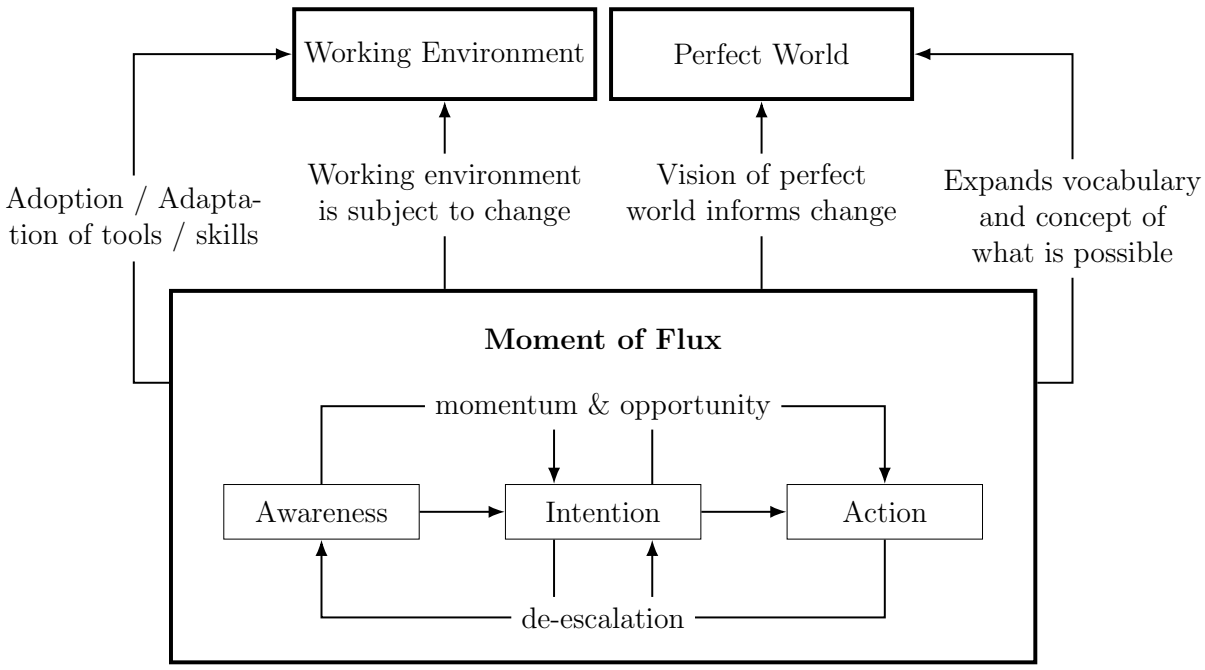

Figure 6.1: **The cycle of deliberate change, zoomed in on the moment of flux.** Building on Figure 1.1 by breaking the "moment of flux" into its components. The flow from awareness, to intention, and finally to action is driven by momentum and opportunity, which can be (often is) external to the individual of project. The flow can also go in the reverse, "de-escalation" direction.

kinds of deliberate changes to the working environment are discussed. The times when they are painless become notable, and there is an expectation of unpredictability in terms of how much work would reveal itself as necessary. More worrisome for many people, it is not clear until it is too late what *kind* of work and whether it is in their repertoire. The **moment of flux**, here, is that moment when the researcher sees an opportunity for this kind of "plunge." Though attending a talk in the example above may seem relatively involved, attending talks to maintain awareness and connection to best methods is typical in an academic research setting. The decision to attend a talk about particular talk is therefore precisely the kind of small, mundane, and important decision-making that is the focus of this chapter. As in the rest of he manuscript, this chapter addresses programming skills specifically, although skill acquisition more broadly is relevant to all aspects of scientific work.

The story of "taking the plunge" above is an example of deliberate change which involves awareness-building. It may escalate from *awareness* of an issue and possible solutions to holding the *intention* to pursue one of those solutions. It rises to *action*, in what I refer to as the **moment of flux**, only when the opportunity and momentum for pursuing a change coincides with the intention. Figure 6.1 illustrates the relationships between the concepts, with awareness, intention, and action all part of an increasing buildup of the desire to change, and the moment of flux itself being composed of (internal) momentum and (external) opportunity. This figure reflects "zooming in" to the change part of the overall cycle, as is illustrated in Figure 1.1.

6.1 Taking the Plunge

One of the 20 participants explains their motivation for attending the SWC event (emphases and numbering-marks added), quoted below, with emphases added to identify components (bold-faced and numbered) that exemplify the ideas in the rest of this chapter:

> Ive been doing increasing amounts of data processing, starting during my PhD with a lot of my own data, using some larger online databases. I am **planning on**

> **starting doing more of that (1)**. I guess there's two things, one is that often we're just learning on our own, and thats effective to a certain extent but if you ever want to try something new, there's a lot of **inertia for trying something totally new (2)**, so it's nice to get an introduction into whatever's going to be new to you, to give you that boost so that you can actually not be afraid to try it. In my case, it was GitHub that **I'd heard a lot about (3)**, didn't really quite understand how it worked or what its purpose was, so I wanted to learn about that. The second thing is that I'm starting a lab of my own as a professor in the fall. And with my own students and postdocs coming in, I want us to do things like GitHub and version control. [I want to learn these skills myself] so that I can **best teach them and establish good practices within my own lab (4)**. – P02

The participant had been "doing an increasing amount of data processing" (programmatically, involving code work) and is "planning on starting doing more" (1, in the quote above). This pertains to an anticipation of the near-or-far future (a la 'anticipation work' [60]), and is an example of future-oriented motivation for programming skill expansion or acquisition. Points (2-4) exemplify typical sentiment toward GitHub as an instantiation of a desirable best practice. Because the speaker places trust in the social context (3), it is not necessary to "quite understand how it worked or what its purpose was" in order to attend a workshop. Attending the workshop *is* the act of change that is deliberately undertaken.

How long is it necessary to put time and energy into something before giving up on it? The kind of of change this chapter focuses on requires a step into the unknown. Code work is associated with an inability to foresee the effects of change at every level of granularity. Figure 6.1 shows the process in deciding to make a deliberate change, in which awareness escalates into intention and then finally into action. This diagram illustrates how the working environment and the perfect world collude in the act of deliberate change, and acknowledges that embarking on change may not always result in follow-through, let alone adoption. Sec-

tion 6.2 focuses on the awareness-intention part of this flow, whereas here I focus on action, momentum, and opportunity.

Momentum is built through community interest and through awe, curiosity, and skepticism that push an individual to seek out alternative methods. Opportunity can be in the form of mandatory policies or requirements, or in limited-time events. Additionally, new project energy as well as breakdown of existing process provide both opportunity and momentum for change. For example, when a database error results in a post-doc in the CustomInstrument-Lab group having to re-ingest all her data into the database, this also presents an opportunity to address some formatting and documentation shortcomings of the past. The opportunity is, nevertheless, a silver lining on an otherwise "devastating" event that did not result in data loss but *did* set the researcher back a month (her estimate, in an informal interview context, as she recounts the narrative) and led to a decision to switch to smaller-scale analyses that did not require "big data" management or processing infrastructure to the previous extent.

Breakdowns need not be severe to inspire change: they may just bring to light the elements of the current practice that demand time but are not an *investment* of time. In the Omics-Lab, the typical process that involves Excel is not only painfully time-consuming for larger datasets, but the time spent further underlines how much *more* time will be spent on this in the future and how unscalable the approach is. Writing a script, on the other hand, involves time-consuming learning, but this is an investment. Mallory from the CustomInstrument-Lab noted a similar attitude about learning to make publication-ready charts using Python rather than MATLAB: it would have been faster in MATLAB, but this was an "investment" for the future. This was also prompted by her MATLAB license expiring, and the realization that the monetary cost of maintaining it would not be an investment.

Statements like, "I could do this all [day/week/month], I could let it take up that time, but I'll just see how far I get in [a more limited span] and that's that" are common. Taken together, they identify self-imposed or external time limits as helpful in the decision to try something new. The ability to stick to these self-imposed limits varies person to person.

A workshop event provides structure (opportunity) for somethingfor which there is already enough interest. Events provide structure in terms of time, as well as additional structure in terms of curriculum and helpers. The mere *existence* of this structure is an important benefit the event: the investment becomes more straightforward, and leap more accessible.

The decision to pursue change requires guidance from a trusted individual on the *realistic* costs and benefits of *concrete* alternatives(s). This trusted individual plays the *software advisor* role by *championing*[2]. The champion can deliberately work to not underestimate cost, by learning more about current working environment; as well as work to not overestimate benefit, by learning more about other available options. Ed, the Software Advisor from the RegionalNowcast-Model team, exercises thoughtful caution when it comes to advice, focusing on what is more necessary at the time.

The outcomes of such a "moment of flux" is **not necessarily adoption.** This is the core thesis: going through the process of informed attempt and subsequent rejection of a particular tool, skill, or process has a noticeable impact on the individual and collective imagination. If we were to enumerate the possible outcomes of moments of flux, we could identify *any decision* (both to use and to not use) as qualitatively better than becoming overwhelmed, retreating from the attempt to incorporate the tool, and persisting with the existing environment out of inertia. This latter form involves paying at least some of the time and energy cost of commitment, but without the benefit of greater control and with less sense of being subjected to an unnecessarily painful process from which "everyone" else has moved on.

Change to a working environment includes (1) *integration* challenges, both in terms of the daily work of the individual and figuring out how to share things with colleagues and collaborators. Aside from figuring out how the working environment must change to

[2]This new role label is used only in this section, and introduced because the "chapmion" if a particular tool need not be the same person who typically plays the *software advisor* role; it can jut as well be a visitor giving an envigorating talk about how scientists can implement better programming practices, or a perceived expert who expresses enthusiasm about something in her audience's awareness, escalating it to intention (see Section 6.2)

incorporate the new tool or skill moving forward, the scientist must also decide on the (2) *extent of migration* of older projects. There is a cost in maintaining multiple working environments, both in terms of attention and compatibility of small utilities. Migration may especially affect changes that take on additional meaning besides functionality. For example, SWC Interviewees described the need to make code "prettier' before putting it online as a barrier to migrating old projects to GitHub. This barrier is great enough to keep GitHub in the scientist's awareness and intention, not escalating to action.

There are also issues of **scale** and **centrality** that become salient in the fragmented and seasonal context of scientific code work. Even when there is a large software component a researcher works on, there are also many other projects, to most of which the scientist not only contributes, but leads or "owns". As such, the burden of management falls on the scientist, including the responsibility to decide how to divide up the components of her work into meaningful but manageable parcels. The scientist typically has a variety of "projects:" a project from their previous work (e.g., their dissertation project), which perhaps has been picked up by a student or colleague; at least one "side project," following up intermittently on interesting ideas which pique attention and await fruition sometimes for years; and any ongoing collaborations. Most people keep such projects separate, as a project has associated with it data files, visualizations, code for the visualizations (an intermediate visualization, if the "side project" blooms to fruition, will to be re-made with the correct formatting for publication, or with changes), notes about analysis, and so on. The "centrality" of the resource depends on the interconnectedness. For example, though some data is intended to be available for a while, groups may choose to have back up data on their servers "just in case," "for posterity," or to be aware of group members who had downloaded data in the past. The members of the BioGeoChem-Model group both worked primarily on their own, and maintained a number of ongoing connections.

From the observer's standpoint, this unsatisfying type of non-adoption is not distinguishable from the satisfying and meaningful one without asking about it directly. For example, in the later section 6.4, I tell the story of the Omics-Lab deciding to try out Tableau. In the

conclusion, an R implementation of a particular chart is preferable. However, the reasons for this preference included a studied rejection of Tableau for R, based on its capabilities. Even following a completed and/or successful "leap," when the once-oblique tool becomes readily usable or the once-obscure practice turns into second nature, it would be a mis-characterization to say that the "problem" that precipitated the action was "solved." Even if the initial prompt was a problem and the championed solution addressed it, the goal of something like a "best practice" is sustained impact. Nevertheless, this itself is distinct from sustained use of the particular intervening solution.

The outcome of a "leap" cannot be assessed by measuring adoption alone. The best possible outcome is an informed decision, which *may* be the decision that the tool or protocol in question is inappropriate relative to the problem. A worse outcome is overwhelm-based inertia: an unresolved sense of "we should use this." The worst outcome involves paying the time and energy cost of committing to some solution, but without the benefit of more control. Attempted-adoption, resulting in an informed rejection, is a positive outcome because adoption requires learning a skill (installation and integration pertinent to the tool in question) and a language, which is invaluable for gaining access to additional resources. In the words of a SWC Interviewee: "even writing down the question or the issue, formalizing that into a sentence that you can understand, and then thinking of search keywords, and then visiting those resources or talking to other students." The ability to articulate requests for help, as reviewed in Section 4.1, is a major resource at the disposal of the scientist in a problem-solving situation.

The day-to-day reality of scientific work is a series of choices made in an attention-sparse environment. Adopting tools or social protocols that are already in the awareness-intention pathway is met less with resistance to change, but more with overwhelm due to an embarrassment of riches. In this case, the task of the champion is less to persuade and more to avoid misleading or mis-representing, as this could quickly exhaust an already overwhelmed audience, thus making that audience less receptive.

6.2 Awareness and Intention

The high cost of follow-through on some call to action, in terms of attention, time, and money, means that for the most part, ideas about change remain in decision-making limbo. In the previous section, I wrote about the *action* step of the *moment of flux*, as illustrated in the Figure 6.1. Escalation toward action requires *opportunity and momentum*. In this section, I also review *de-escalation*, and the pre-cursor steps of *awareness* and *intention*. The primary difference between these two "steps" is that neither involves actual adoption, but one is much more receptive to suggestions from a trustworthy "champion" of a tool or protocol.

Awareness is amplified and sustained through social channels ("I had heard about git but I havent used it"). As a scientist continually evaluates and re-evaluates their current working environment and adapts their course of its intentional and unintentional change to point toward the collective imagination of the perfect world, he may at one point form an **intention** to make some particular change or learn a particular skill. This intention follows awareness but may remain passive indefinitely (e.g., "Python has been on my to-do list for a long time").

Intention can subside down into the maintenance of background awareness, when it is not driven forward by external or internal opportunity and momentum. For example, in one of the post-SWC interviews a participant tells me that she "went looking for best practices" but "life got in the way" and she needed to do more research design instead of scripting. Ultimately, she used scripts (new knowledge) for selecting sites for data collection, a way in which research design ended up creating an opportunity. However, other higher-priority concerns (professionally or personally) "getting in the way" is an example of intention *not* leading to action, but instead retreating back to awareness.

The moment-of-flux relies on not only the momentum and opportunity, but the awareness of possibilities, which is maintained passively over time in the normal course of doing science. For example, Figure 3.2 illustrates different types events observed during this work. These

events are arranged by how much close interaction occurs between the participants/attendees, and how targeted the agenda of the events is to solving a particular problem. Those that are less targeted help to build awareness of the various options, as well as develop the language around concept. This *awareness-building* is expressed then in statements like, "I had heard about git but I haven't used it." Having this awareness, and seeking out its maintenance relative to a changing landscape of available options, is built in to the range of targeted-ness of group activities undertaken by scientists.

In my first observation of the RegionalNowcast-Model team, I met Ed, the *software advisor* to the group. Over the course of several months in this study, he went from advising the group in after-hours meetings, to being a formal part-time employee. The first time I met him was at an hour-long meeting from 5 to 6 pm, and after it was over he told me a bit about his background, motivation, and approach. I would summarize a big part of his approach as strategic omission of unnecessary detail, where he has to make deliberate effort to determine *which* is the detail that is unnecessary in the context: "you know I just watch them, I can see when their eyes glaze over." The relationship he and the PI of the group have had, which was a motivation for him to have gotten engaged in the group prior to having a formal position, also allowed for the kind of feedback that would support this. He tells me, later, that the PI is "really good at pushing me on it," asking him: "do I [as a scientist] really need to know this?"

The question of how much understanding is needed for a sense of comfort and control depends on the context. Mostly the participants in the study wanted more detail, not less, from software. However, even in the above example, the PI did write code in the shared project and run analyses. The "do I really need to know this?" was asked as an honest question of what is actually necessary to have a *workable mental model*. One of the informants was recounting having worked with a computing researcher who was specifically aiming to help the group, but it became frustrating: "I want salt, but he comes back a week later with a machine that not only gives me salt but pepper as well!" Sometimes the effort to make something "usable" seems to invite flexibility, feature creep, and *less* understanding

or control over a *more* complex product (Section 7.3 tells a story of one such criticism). The irony in this case was that the function that the researcher wrote was in response to an R question, but there was already a simple one-line existing solution, "but the guy didn't want to take the time to look for it." The telling of this story relies on the wisdom of hindsight: the speaker relates this telling after having learned enough R to have found the function.

Few people were still using any of the specific skills they had learned at SWC afterwards. Some had additional motivation embedded in their immediate social context: "my whole lab uses this, so I have to" or "I am a new faculty and I am starting my own lab, and I want us to use best practices." Some had de-escalated from *action* back to *intention*. For exampleas I had interviewed people multiple times over the course of several months, I could hear the progression from (1 month from SWC) checking out a book from the library on a visualization package in R to (2 months) not quite remembering the book at first when I ask about it to (3 months) just wrapping up with a major task, and looking forward to finally getting into the book. Such disrupted time-lines are not the exception in academic work.

6.3 The Exciting Thing

Notably absent from my account of decision-making are the social, financial, or institutional pressures and contexts that are undeniably important to shaping actual decisions. Issues of resources and power dynamics are all included under the "opportunity" umbrella, despite the extent of their centrality in shaping what work a particular researcher does. Additionally, evaluation of utility - such as evaluations of the best course of action to solve a given problem - are not directly present in the model, remaining implicit in the awareness-intention-action sequence. The core of the escalation from *awareness* to *intention* to *action*, and the follow-through in the *moment of flux*, as I have painted it, is emotional engagement and excitement.

There is a particular room on campus in which two of the groups had independent joint meetings regarding visualization. The two meetings were not related to one another, other than their broad goals to include some computer scientists and to talk about visualization. The room has 2 walls with project screens opposite each other, a long window that includes

an expanse of sky and part of the nearby marina, and a long glass wall with doors dividing it from the rest of the office space. This is part of the Moore/Sloan Data Science Environment on the UW campus. The glass wall includes a sliding door, which people keep trying to push and pull, so it often sports a post-it roughly at eye level to remind them of the proper usage. In both of the visualization meetings, the projector is used to show examples of visualization in order to structure forward-looking reflection. In both of the meetings, few people had laptops, unless they were actively projecting, or unless they were from the computer science department.

In one of the joint visualization meetings, members of the BioGeoChem-Model group met with members of a particular computer science group specializing in visualization research. The meeting was by way of introduction, and had been organized by Mary, a post-doc in the BioGeoChem-Model group who was also engaging with the computer science groups through regular meetings. Earlier, Mary articulated the reason for the joint meeting to me as being "to establish lines of communication" and further went on to note that "they seem interested." If we were to place this meeting on the 'targtedness' scale of Figure 3.2, it would be on the low end: the goal was to build awareness and establish the foundation that would be necessary for future engagement. The format of the meeting, therefore, was for all the oceanographers to show a chart, which would accomplish two goals: to explain what they work on, but also to get critique. Because of scheduling constraints, this meeting had been in the air for weeks, and a week before, Mary seems worried, noting that "there was a scientist presenting at the last meeting and I dunno... they just seemed kinda... bored."

The joint visualization meeting for the CustomInstrument-Lab included the typical members of their regular, approximately monthly meetings, including Melissa. She is a computer science graduate student who has been working, on and off, on data analysis and visualization within the umbrella of the CustomIstrument-Lab. Even more than usual, the physical space was self-segregated, with the computer science and eScience members of the collaboration on one side of the elongated conference table, and with the oceanographers on the other. Melissa is one of the few people with a laptop, and she is sitting across the table from

the PI, who as always lends a bright-eyed enthusiasm to the meeting. Half an hour in, she addresses Melissa directly, "I haven't heard as much from you as I have from others," and encourages her to think of some things she would be "excited" to do. Melissa replies: "I am listening for the use cases and needs," adding that she "would be interested in redesigning" the website where the group currently displays their data.

The website is an important analytic tool as well as a means to showcase their work for stakeholders. In terms of data stewardship, it does have the formal burden of making data available in a particular way demanded by the grant, but there is also the desire to make the access *useful* and *cool*, taking it beyond something that 'only' has the minimum functionality. In the meeting, Melissa takes over the projector and navigates to various example data endpoints from other ocean research groups. Some have drop-down menus and forms to select a part of their data to download, some have dynamically-generated visualizations. Others in the meeting, most notable Mallory, George, and Andrew, chime in on features that are notably good or bad. Seeing an overview is good, like showing all the cruises simultaneously on a map. Zooming and filtering is good, like being able to select by geographical region or by time.

The PI suggests a particular website as an example of "really nice data" and a "pretty landing page." Projected on the screen, it is a cornucopia of visual content. There is a world map projection - the kind that is a single stretched oval with the Pacific in the center and with a modestly-sized Greenland. Below, there is an animation of how a particular float works, super-imposed on a photograph of the surface of the sea. In a prior interview, Melissa had mentioned her experience in trying to work with Argo data; this story is included earlier in Section 5.1.

On the site for downloading float data, Melissa pauses briefly, looking for where in this she can download the data. It turns out that a few clicks away from the landing page, there is another page with a plain background, bullet points, and no special formatting for any of the many longs links included in those lists. There is a little joking (the PI quips, "oh this is done in.. 1997?" and then, turning to Mallory: "is that your derogatory year [for

outdated-looking websites]?") Melissa does not engage and instead focuses on exploring the different capabilities of the interface. When Andrew begins to offer technical opinions and detailed ideas for next steps, she responds. Their shared background and opinions contribute to a constructive resonance in a quick volley from which materializes a plan of action. The plan of action involves a meeting between the two of them to work on the endpoint in a few days.

Unlike the BioGeoChem-Model meeting, this one is much more targeted. There is a concrete artifact, and there is a variety of issues in the shared *awareness*. In particular, there is the recurring challenge of whether the endpoint needs to be different for archive data versus real-time data. Aside from different use-cases, there is also the challenge of data storage, and at which point what needs to be stored where. In the working meeting between Andrew and Melissa a few days later, they get a few small tasks done, help each other understand some code, and overall work on the assumption of maintaining as much unity as possible. Months later, Andrew makes a decision to split the storage from the endpoint, as a result of reliability and speed issues with the unifying data storage component that had been used so far. In this case, the intention to split up the components had arisen every so often from recurring updates to the awareness of various options, but it did not transition to action until a breakdown created an opportunity.

By the time the decision is made, Melissa is no longer working with the group. The visualization meeting followed about a year of uncertainty regarding the nature of research excitement for her. In a prior interview, she told me that "I just want to do something that's interesting and I will figure out later if it makes sense [with my graduate degree requirements and expectations]." David, one of the other members of the eScience part of the broader CustomInstrument-Lab collaboration, had approached her, "we need to switch your project, don't we? Because you're not enjoying it," to which she says she responded, "Yes, I'm so glad you noticed that because if you hadn't said that this would have not gone anywhere. I would not enjoy it, but I could do it," in reference to a previous project not with the oceanographers. Of that project, she noted that: "I go in to it with high energy, the meeting is reasonably

high energy, I'm excited about the things I'm talking about, then somehow it peters out at the end." As she tells me her reasons for wanting to collaborate with oceanographers, it is clear she is interested in the working with the data and understanding the science, not "just" engaged in the service role of tool construction.

In Chapter 2, I aimed to paint a picture of oceanography work as pluralistic and inspired; there are many ways of knowing, and each approach is individual to the group that pursues it. This is not to say that all endeavor in the discipline is sustained by individual passions of curious post-docs and graduate students following their hunch and encouraged by enthusiastic PIs. Truex *et al* offer a discussion of what they call 'marginalized' code practices, such as extensive copy-pasting, as a counterpoint to the 'dominant' view of code work that emphasizes design and 'best practices. This is not done to claim that the marginalized coding approaches are any more common than the dominant, but to provide an alternative narrative that includes practices that *do exist* not as an outlier or aside, but the primary focus [64]. Similarly, my aim in centering the narrative of decision-making on curiosity and awe is a way to re-cast a conversation that is about problem-solving (e.g., [29]) or struggling against many competing forces (e.g., [60, 54]).

The world without the most obvious pressures, like money and recognition, would be a fictional one. Nevertheless, this fiction reflects a certain imagined world, and contain a kernel of truth revealed in the intentional separation of these external, practical matters from the craft of scientific work. When the Omics-Lab group takes time to discuss the cost of equipment, it is an exchange kept to a minimum in terms of its demand on time. When the PI of the RegionalNowcast-Model group talks about her tasks for the day, she delineates the times spent on grant-related activities, which must be done before she gets to do the "fun part" of data analysis. The "fun part" and "playing with [data or code]" refer to the intrinsic joy of work, and particularly code work since that what I was observing. The PI of the CustomInstrument-Lab, Andrew explains to me one day over lunch, is in a position *because* of her financial resources to create space for workers like him to pursue more uncertain software projects. The implication is that the freedom she creates lies in the

capacity for the group (to some extent) to make decisions without as much concern. Mallory, talking about a junior student she mentored during some time I was observing, notes that his stipend was coming out of a separate pool of funding which was nice, because then she did not have to worry about it. The struggles with these external factors have that quality of being at the forefront of attention only when there is a scarcity, and there is an effort to limit the extent to which that struggle impacts other aspects of work.

Some, like Erin in the RegionalNowcast-Lab, created spaces during her day for creative uncertainty. At the end of a shadowing session, she comments that she is "pretty happy I got this done... this is one of the bigger things to do today... often something else interesting comes up, which happened this morning, otherwise, kept to task well." Contrast this with the end-of-day reflection of Aaron, a programmer working with the computer scientist collaborators of the CustomInstrument-Lab team: throughout the day, people came to him with troubles, and (among other issues) a dashboard for monitoring database health was buggy, so he did not manage to do any of the things he had intended to do in the beginning of the day. He left tired and in a hurry to catch a not-too-frequent commuter bus home, emailing me from the ride: "I spent way more time than I had hoped debugging the [dashboard visualization] issue, and had no time for coding :(Unfortunately, this isn't unusual..." Things that just "come up" are both unknowable and inevitable, especially with code work, where the working environments have frequent, unexpected brittleness. The moments that create opportunity and momentum arise from such breakdowns as well as from curious follow-up on the "interesting" and the "weird." Though they are influenced by many other factors in the relevant context, the "ideal" navigation through these decisions is imagined to be free from them. More interestingly, this ideal is also, in this setting, full of language of emotional engagement, rather than intentionally-detached utilitarianism that is implied in the oft-repeated idea that "scientists use code as a means to an end."

6.4 Trying Out Tableau

In introducing the *perfect world* in Section 4.2, I mention a time that the Omics-Lab team decided to try out Tableau. In this section, I delve more into this story and demonstrate how to use the concepts introduced in the prior section with a case study of trying out Tableau. Tableau is is a software package for visually exploring data and for creating interactive visualizations in a GUI, drag-and-drop environment. In this case study, the members of the Omics-Lab group are considering use of Tableau. Typically, their work involves Excel to manage and process data, and to produce visualizations, in addition to proprietary software for analysis of mass spec data. In this section, I tell the story of two different members of the group deciding to try out Tableau, which is met with excitement and enthusiasm, but which ultimately prompts one of the group members to use R instead. R is a statistical data analysis language, most commonly used in the RStudio environment, with a number of popular packages for rendering visualizations.

R is distinct from both Excel and Tableau in that it is free and open-source. The other two are proprietary, though in context neither is cost-prohibitive. Tableau is based locally in Seattle, so the PI of the group mentions that she actually happens to know someone at the company, and there could be enthusiasm or opportunity for collaboration in which Tableau could become more useful for supporting the kind of science this group does. It is not the cost but the control and community of R that make it the preferred choice in the end. It allows fine-grained control of features that allow for publication-quality visualizations, such as lines, colors, and fonts. Although it is useful for exploration, Tableau's intermediate visual representations would have to be re-implemented anyway. Additionally, R has an enormous user community that share code and examples online - a benefit, articulated again and again in many contexts of the study, falling in the *on-line/on-demand resource category.*

In an interview with the PI of the Omics-Lab group, she tells me that **(1A) "historically my lab has not done a lot of research that required people to handle large sets of data**. [But with the new projects that the lab is involved in, by virtue of the scope and

nature of their collaborations] we ourselves **(1B) are going to be getting ... this huge volume of data**, and without those kind of skills some programming skills, some database skills, they're not going to be able to get through their degrees. **(1) I tend to forward any kind of email about big data or learning software tools for handling data, visualization ... to let them know that I think this is stuff that they should be paying attention to** ... I know that they're not all going to be able to, but if enough of them do it enough of the time and it will start to catch on and they'll start to meet people and hear about more stuff... I'd like them to just get that exposure. If I don't push them in that direction, I'm being very neglectful." At some point in one of the regular weekly meetings she mentions, offhand: "**(3) we should** use Tableau, **(4) someone was telling me its amazing**".

The added emphases reflect, respectively, the (1A-B) *anticipation* of change in the methods of the group as it is influenced by changes in the broader scientific community, (2) the deliberate effort the PI of this group makes to support building *awareness*, (3) the expression of the *intention*, and (4) the interpersonal persuasion as a factor in elevation of Tableau as a data exploration tool from *awareness* to *intention*. In this section, I explore the elevation of Tableau into *action* in a *moment of flux* enabled by the *opportunity* and *momentum* of analyzing new data by two different members of the group.

The more generalized desire to "push them in that direction," meaning, the direction of learning data science skills, programming skills, or skills useful for working with large datasets, was expressed in the members of the lab attending a SWC event[3].

From an interview with a particular postdoc, PD, on why she went to a software carpentry event: "Yeah, I think our group in general is pretty new to analyzing big sets of data ... it was our first step as a group to learn some of these [tools] together. The other girls have been more successful with using R, for example, which I haven't any time to figure that stuff out. Our group in general is just looking for ways to help us in the future when we

[3]One of the ones I observed, which is how I encountered this group and what prompted me to include them in the study.

get the extractions down and we've analyzed all these samples from the ocean, how do we visualize that? How do we look at what proteins are there and how they're changing and what organisms are dominating and those kinds of things? I'm very new to that, so I was hoping that the Software Carpentry class would kind of help."

Later in the day when the interview takes place, during the meeting, the PI points out heatmaps to us as examples of these awful unreadable things. Later, after also interviewing Colleen, a post doctoral fellow, we set a time when I would shadow her as well as show her how to use Tableau to make a very particular visualization, after processing some really specifically-formatted files in Excel. During the shadowing session, she gave me a detailed look at how their mass spectrometer differed from the mass specs many other adjacent labs were using in terms of (a) what it does and (b) what it outputs. At the end of the day when she got the data she and I spent a little time fidgeting with it in Excel and then in R, because we were trying to convert it from wide to long to get it into Tableau. The following week the PI sends an email that she has a meeting halfway through the scheduled lab meeting time, which is unusual. Colleen replies along the thread: "I *may* have some Tableau graphs to show... And some updates on the [research subject] front. This could also wait until next week where the data processing kinks may be worked out?" To which the PI responds: "I totally want to see that, Colleen! Sadly I may have only a short time to meet. I have to leave by 1pm. I can likely be done with class by 12 since I have to leave during the evaluations. Maybe we can start even earlier, like 12." The meeting was moved to an earlier time so people could look at the plots, and it served to de-mystify Tableau: it was clearly not a total, magical solution to all problems, but it also clearly had notable benefits. As Colleen had previously said in an interview about the SWC event she had attended with two other members of the Omics-Lab group, "It gave me the idea that there's a couple software programs that can be helpful, but it's going to take a while to learn how to use those. I was hoping that it would kind of say, 'Okay, you just have to use this algorithm and it will plot all your proteins for you.' That would be easy, but it's not going to be that easy."

What do we get from using the terminology of the *moment of flux* from this chapter in reasoning about this event? For one, we can see that in suggesting to show something about Tableau I may have inadvertently created the initial opportunity for which momentum had already been building. My action was not intended in this way. However, because I undertook it, I became a kind of "champion" of this tool, despite my not actually intending to advise the team on anything (the impulse to offer help came from a sense I had during that time in the study that I was increasingly a burden, in terms of energy, and should do something useful but not intrusive. Occasional mentions of use cases that could clearly benefit form Tableau, and prior mentions of the software, prompted me to engage in this way). An unintended outcome could have been that when the second group member tried to use Tableau, found it offered her insufficient control, and moved on to R, she seemed almost apologetic to me explaining why she did not like Tableau. Perhaps if I had this model of deliberate change and flux, I would have not created unnecessary tension in an effort to be helpful?

A similar dynamic could be seen in the recurring joint meeting for the CustomInstrument-Lab team that includes both oceanographers and eScience and computer science team member. As all the researchers and staff take turns making a "check-in," explaining what they have been doing since the last meeting a few weeks ago, George says, in an almost apologetic tone, "we modified [the code]..." to which Leonard responds: "you say it like you expect me to be horrified! no! *please*, maintain it!" In that group they also speak in the absence of one of the external (as opposed to in-house) software advisors, Eric, that they should use a particular working environment set-up "because he wants us to use it." Meanwhile, when I interview Eric, he notes that he would prefer it if the members of this group used him as a source of feedback, rather as someone supporting and advocating for a particular tool even if it not the right thing.

6.5 Summary

The **moment of flux** is a short period of time during which the possibility of change is embraced and a few alternatives considered (or even just one), with the outcome of possibly

changing a particular part of the process. A day is full of moments of flux; though they are deliberate individually, and may have cumulative impact, though the discussion in this chapter has been centered on the immediate. The moment-of-flux kind of change is distinct from slow drift. The "slow drift" type of change might be, for example, a different file-system directory organization that copes with the existence of GitHub repositories but which arose on its own, and the distinction of which from the pre-GitHub directory organization seems to only become apparent when an observer (me) asks for an explanation. One example alternative to this is a topic brought up on a meeting of someone checking in a large image into the code repository, which catalyzes a discussion and a self-reflective verdict of where what files ought to go now that everyone is using GitHub. This change that is (1) deliberate, and (2) small but noticeable. It is not the institutional or mandated overhaul of practices, and it is not a slow unintentional drift of practices, but it impacts those processes by way of reflection and occasional refusal to pursue particular directions that are not "exciting."

Whereas in Chapter 4 and 5, I paint a picture of deliberate change as making incremental improvements to the *working environment* in the directions of the *collective imagination of the perfect world*, Chapter 6 zooms into the *moment of flux* and elaborates on the sources of uncertainty, as illustrated in Figure 6.2. The everyday uncertainty of code work, and of the routine integration of new components and skills, is revealed only at a closer look.

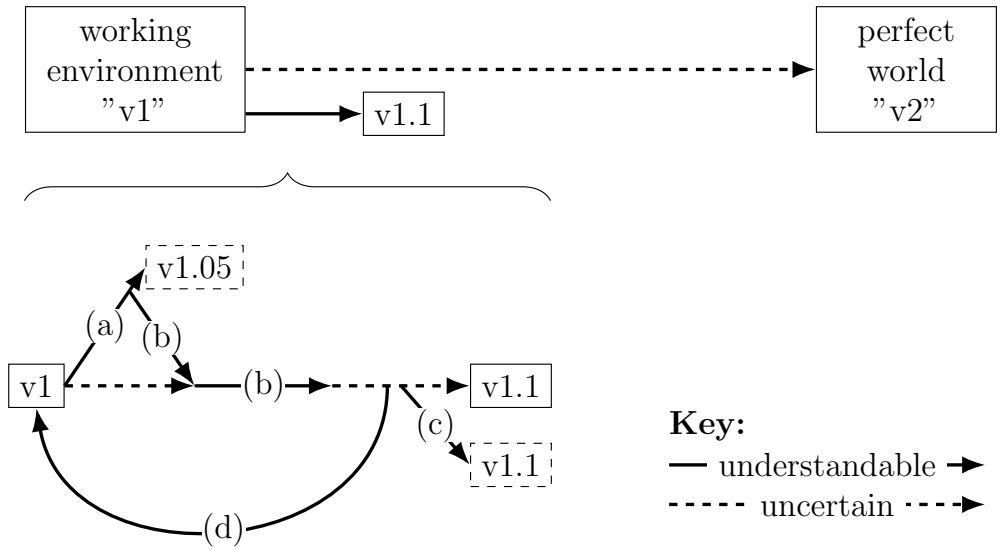

Figure 6.2: **Illustration of incremental change as containing uncertainty, which plays a crucial role in the *moment of flux*.** Although the researcher may have had prior experience with addressing each of the following challenges, they still have few means to estimate when they will come up: (a) integration challenges that demand changing another component first; (b) verification and validation of the overall pipeline post-change; (c) discovery of a preferable alternative; (d) finding out that the intervention does not solve the problem. The figure distinguishes steps that are relatively approachable and map onto prior experience (solid line) from those that are filled with uncertainty in terms of time and commitment (dashed line).

Chapter 7

EVALUATION OF INTERVENTION OUTCOMES

The expectation of an experimental software product to be the primary solution to a major problem does a disservice to both creators and users. This chapter explores alternatives to the narrative of software interventions that paint a promise, identify an associated obstacle, and propose a solution in the form of a software intervention. Figure 7.1 illustrates the different intervention contexts. Each of the sections maps to an intervention in a different context, and concludes with a take-away that follows from the conceptual framework developed in the prior chapters. These are summarized in the final section as encouraging intentional recognition of ways in which groups make progress even when specific technological interventions do not work out.

An over-zealous emphasis on the novelty and centrality of particular technical challenges (e.g., big data) is both historically erroneous in painting scientists as passive recipients of qualitative shifts in available technologies, despite the critical role of these very "end-users" in having brought about these shifts in the first place, over much larger time scales than some technology-innovation-centric abstracts would suggest. Computer-science study of scientific work sometimes describes the current state of programming, code, data and analysis infrastructure, and software production in general (in the scientific domain) as somewhere between unsustainably frustrating and untenably chaotic. In an early 2006 writing regarding the Software Carpentry curriculum, Wilson writes in the introduction that only few participants in a recent course used version control or any kind of code-checking tools, noting that "I wish this were unusual but it's not... [but] it doesn't have to be like this" [68]. The explicit implication is that not testing the software produces scientific errors, which is indeed a major

challenge. However, sweeping dismay over non-adoption of best practices, like testing, may fail to acknowledge effective utilization of alternative systematic forms [17].

Recent technological advances have changed the set of available tools, accepted practices, and institutional resources. However, what is 'recent?' A mid-2000s article speaks of a 'data avalanche' in environmental sciences, calling for a "computer science agenda spanning databases, visualization, workflows" [29]. Speaking about the early 1950s, Edwards writes in *A Vast Machine* that "climatologists ... faced not only an explosion of data but also a plethora of alternatives for coping with it" [18]. These are by far not the only paragraphs on the topic that paint a picture of the scientist coping with "cataclysmic" amounts of data, requiring technological innovation.

This narrative does a disservice to both computer scientists and domain scientist working on cutting-edge experimental software. It fails to use the rich resources available from the domain scientists side, and it fails to provide room to recognize the successes of projects that do not result in a complete achievement of the technical goals. In a landscape that benefits from experimentation and trial-and-error, I argue that there are more productive alternatives. Each section highlights an alternative, and the concluding section synthesizes these into an evaluative stance that emphasizes social and skill outcomes for building and sustaining constructive momentum.

First, this view **underutilizes the technical, social, and cognitive resources of domain scientists.** In the prior chapters, I described the working environment of the oceanographers as including not only particular tools (iPyNB, MATLAB, GitHub), but also social resources ("documentation" over email with colleagues) and cognitive resources (extensive visualization vocabulary used for validation and verification; see Section 4.3). The negative outcome of this under-utilization is the focus of Section 7.1, in the context of an experimental component of a large infrastructural effort that corresponds to (a) in Figure 7.1.

Second, this narrative **suggests a very limited success expectations that do not include other forms of impact.** Such an approach of finding problems, or opportunities,

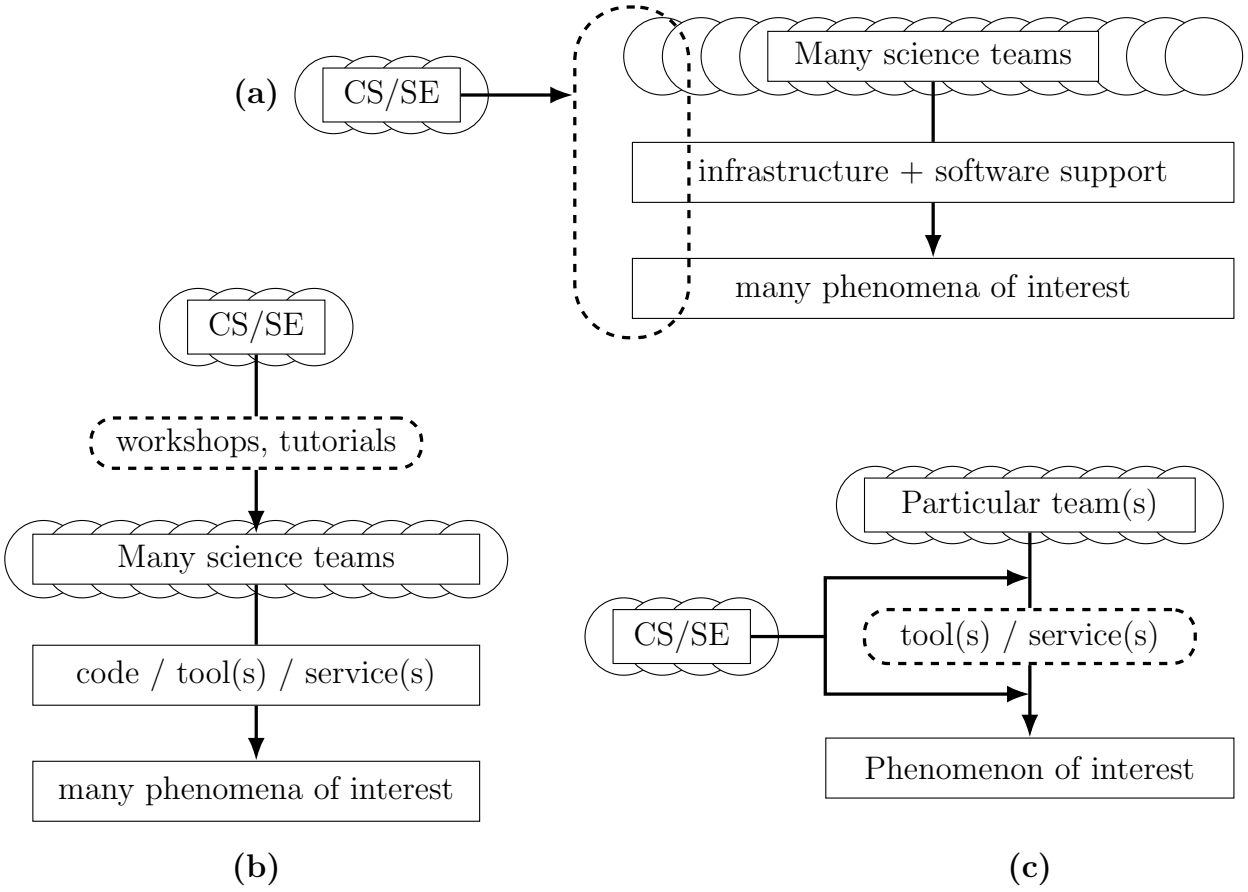

Figure 7.1: **Three different ways in which computer science intervenes in domain-science code work.** (a) By working on a particular project that is part of a larger infrastructural and collaborative effort and which ultimately aims to support many scientists in their study of many different phenomena; (b) by organizing educational interventions, like workshops, tutorials, and mentorship programs to support scientists who code in developing programming skills, who are then mostly on their own to apply those skills to particular projects and research questions of interest; and (c) by collaborating closely with a group pursuing particular research questions, and building tools or intervention specifically tailored to that group or context. "CS/SE" stands for "Computer Science / Software Engineering" but extends other computing-affiliated researchers.

and building technological interventions to address them leaves few options for evaluation: there is one way to succeed, and many ways to "fail." The "failure" may still offer valuable insights, but the evolution of the problem or the (informative) shortcomings of the solution become defensive arguments. The conceptual framework offered thus far in this dissertation is intended to complement this. Any new software tool or programming skill is adopted into - and adapts - a rich working environment that includes technological, social, and cognitive resources. Even if the adoption is not sustained over time, the attempt itself can translate into an *expanded imagination of possibilities*, if not actual change in practice. Although the sense of elegance of a scientist who codes may not coincide with the sense of elegance nurtured in computer science training and professional communities, it nevertheless informs reactions like excitement and boredom that affect decision making outside of interpersonal or utilitarian concerns (see Chapter 6). Section 7.2 and 7.3 offers stories from my fieldwork that demonstrate more inclusive articulation of goals and recognition of successes.

Third, the promise-obstacle-solution narrative in the experimental, transformative code work context in science **enables maintaining misaligned imaginations**. This, too, is the subject of Section 7.1, where there is a gap between how different *communities of practice* involved in a major interdisciplinary collaboration assign value to different components of the system as crucial to get right, versus incidental. In this following section, I illustrate my claims with a reading of a blog post and a whitepaper about the Ocean Observatories Initiative (OOI). The blog post, written by a prominent oceanographer, criticises the unavailability of the data generated by an impressive hardware infrastructure simply because an unnecessarily-complex software component is behind schedule. the whitepaper, written by the technologists behind this "cyberinfrastructure" component, stress the that the integration only possible through the completion of this component is strictly necessary for effective use of the data. Through integration allows data operations that are otherwise not possible, there is still a notable difference in what the two documents hold as their vision of the *perfect world*.

7.1 The Social and Cognitive Resources of the Scientific Community

"It is MATLAB for people who don't know how to use MATLAB," someone quipped an intentionally scathing insult about a software component of a major collaborative infrastructure project which had become the center of criticism that week. After the harsh aside followed a quick revision that defended the major, inter-disciplinary, multi-institution project as a whole but did not redeem the cyber-infrastructure component in particular. I was then directed to a very long blog post written by a high-profile faculty member of the department, which focused on criticising shortcomings of the project in engaging the resources of the scientific community, and calling that community to more proactive action.

The Ocean Observatories Initiative (OOI) aims to: "to build, in part through computational means, a form and scale of collaboration that would change the nature of ecological and oceanographic work" with implications for "practice, training, policy" as well as "core vocational identities that shape and define research as a way of life" [60]. The blog post, written by "WW," begins by rehashing the timeline and recent delays in the project.

> In about 6 months the [NSF] will complete the construction of the [OOI], an ambitious plan to build infrastructure for long-term studies of the oceans on coastal, regional and global scales. The physical portion of infrastructure will be built on budget and nearly on time... However beneath this veneer of success lies a project that is in disarray... It is probably not surprising that the engineers working [on the cyberinfrastructure component] within the OOI's culture of isolation from scientific users, managed to burn through $30M trying unsuccessfully to turn this grandiose vision into some useful software and hardware. – *WW* [67]

The conflict described in this case study is included to represent a common tension in cyberinfrastructure development [54]. In the take-away for evaluation at the end of the section, I contrast the **vision of technical modularity and extensibility** as necessary for creating an environment of sustained innovation with the alternative view of the

capacity of the cognitive and social resources of the working envrionment to "manually" connect components. A chapter about evaluating success of code work in oceanography is certainly incomplete without considering this passionate indictment of a major, expensive, and high-profile project. The reader is directed to the work of Steinhardt and Jackson for more on the fascinating story of the OOI (e.g., [60]). In this section, I use a blog post to highlight and unpack a particular kind of indictment of large software effort. The author of the post, William Wilcock (WW), offers a call to arms for the scientific community (of oceanographers), beginning with the title "Time to Speak Up" and ending with a list of six key takeaways aimed at the OOI administrative body. The rest of this section will take a close look at this call to arms. ûse prior to intro OOI

The infrastructure, comprising both hardware and software components, make this project both a scientific and a technological endeavor. Lee and Bietz focused on the challenges of cyberinfrastructure design and development. In a recent summary of the adaptations necessary "in the face of changing science" [9], Bietz and Lee highlight the tension of creating contextualized, tailored software solutions against the maintenance and design of a long-lived construction. In a similar fashion that foregrounds software construction and data use in the study of scientific collaborations, Ribes and Finholt identify as a major tension in cyberinfrastructure development as the one between "respecting current work practices" and "transforming scientific practice" [54]: "...when asked about a future trajectory of technological development, [scientists] respond first with their everyday computing pains, and if pushed will generate digital utopias. ... They do not reside at the forefront of computer and are unable to articulate realistic novel applications." Aside from shortcomings in communication and/or imagination, what other explanations are plausible for this common complaint about this kind of interdisciplinary projects?

WW's attempt to galvanize an oceanographer community takes a nuanced view of technology, distinguishing the hardware data-collection components as extending from the oceanographic community, set against the cyberinfrastructure component as unilaterally established by the technologist stakeholders.

The post contains multiple references[1] to a "community" of scientists in oceanography that were left out of decision making: "For a while ongoing planning for the OOI involved a somewhat unwieldy collection of >80 enthusiastic scientists and engineers from throughout the community who were organized into numerous groups aligned with various subcomponents of the program. Rather than streamlining this expertise into three active advisory and oversight committees for the three IOs, NSF rather unceremoniously discarded these groups." WW goes on to enumerate ways in which the NSF could have gone a more constructive, reparative route relative to the scientific community, but did not take these options. This dismissal, ultimately, neglects using the social and intellectual resources within a community that has been working on sharing data through electronic means for decades [18, 36]. Another issue in the post is the neglectful lack of sharing data. WW notes that "community pressure" caused the NSF to agree to make some of the data available ("if this can be accomplished at minimal cost"). The values suggested count *both* ignoring the express wishes of the broader community and leaving "most of the data gathering dust" as major problems.

In a a 2007 description of OOI components published in a conference proceedings, there is also a singular reference to the scientific community. The 2007 paper is one of a series of a dozen technical reports; meeting summaries; white papers; and conference publications spanning 2004 to the present. These documents are rather similar in their articulation of big-picture goals, offering primarily updates; I chose this one because WW referenced it in his blog post to highlight the unnecessary and undesirable feature creep of the cyberinfrastructure. In its concluding section, its six inter-disciplinary co-authors underline the unity, coherence, and integration as the "single most important characteristic," stating that "the ability of any participant to contribute to the overall functionality enables the community to recognize and adopt innovations as they emerge" [3]. WW, meanwhile, criticises the holism and entanglement of the disparate, in his view, OOI components: "NSF encouraged the development of a poorly defined, but all encompassing cyberinfrastructure component

[1] 17 words out of 2.2K are "community," only one referring to a community of microorganisms. It is the most common term after "NSF," "OOI," "data," "observatory," "cabled," and "UW."

that would somehow link the disparate elements into a coherent observatory or 'system of systems"' [67].

Who is right? What level of integration is necessary, desired, or even possible? Is is possible to comment on whether the integration was itself a bad idea, given the major shortcomings on the part of planning and communication? In the wake of a perceived failure, it is difficult to distinguish whether the *concept* was at fault, its *implementation*, or something altogether separate? Moving forward, will large integration or standardization efforts be approached with more scrutiny, and what will that mean for the level of freedom, and the level on anxious uncertainty, in code work components that are either invisible or badly broken?

WW's blog post includes also statements of pride, celebration, and a note of unfettered positivity with regard to the data-collection hardware. The sections underlined further illustrate the frustration and disappointment of being unable to access this celebration-worthy data as a result of "waiting for the cyberinfrastructure to catch up." This frustration reveals the division between the two different kinds of technology: the hardware that the community ought to be proud of, the software that is blocking the progress of the community. The blog post concludes with a 6-point todo-list for the OOI and for "the community," mostly calling for reallocation of resources to the right technology, with wording that elevates pragmatic, clever uses of existing community resources over the "holistic," "end-to-end," "framework" software at the heart of OOIs cyberinfrastructure vision [3].

Take-Away for Evaluation

One of the biggest challenges with larger, more integrated software is how all-or-nothing it can be. Moreover, in software projects that aim to transform scientific work, the technological value proposition and the design are not necessarily known at the outset. In the preamble of one of the joint meetings observed with the CustomInstrument-Lab group, a squiggly graph in the corner of the whiteboard shows a plot with 2 axes and a squiggly line, looping around on itself in an impossible way, so someone idly jokes: "does that mean time went

backwards?!' To which George, the post-doc running the meeting, jokes while he erases the board, along with drawing: "with [our instrument], *anything* is possible!" The language of eScience stresses transformation, innovation, and a challenging of assumed limitations.

The conflict regarding the OOI cyberinfrastructure is only one case, and the blog post is only one person's opinion. I encountered it in context of study informants concurring with the points in the blog post, nevertheless, the purpose of the case study is to use the conceptual framework developed in the prior chapters to offer an additional perspective for the evaluation of interventions. For infrastructural intervention, the tension between the pragmatism (of the domain scientists) and the technological idealism (of the computer scientists) is nothing new [54]. Both narratives - the vision of the OOI from the associated white papers and the articulation of failures from WW's perspective - are about creating an environment where innovation can take place and include a wide variety of scientists and stakeholders. However, the former envisions this as enabled by a technical modularity and extensibility[2], whereas the latter *implicitly assumes* that the field already has the social and technical wherewithal to to cope with the unconnected components that are at the heart of the CI.

There are major, transformative benefits to encoding extensibility in formal abstractions and protocols; it is scalable to more people, while requiring less *extra work*[3]. However, this promise of the perfect world (Figure 4.1, in Section 4.3) also needs a way for individual participants to make incremental changes in their working environment, which spans social, cognitive, and technical resources as a whole even if the intervention in question is primarily technical in nature. Furthermore, scientists who program tend to gravitate toward tools with rich communities[4], so adoption is only improved if a large intervention is made available piecewise, before it is able to support integration, as well. **When building something**

[2]Here, capacity to be incrementally built upon or extended with meaningful but distinct modules.

[3]The additional work of support and maintenance that builders of scientific tools must do to maintain the user community, see Trainer et al. [62]

[4]See "guiding resources" in Section 4.1, as well as the findings from a large survey sample by Hannay et al. [26]

integrative or holistic, consider ways that its components can be made available piecewise and engage with the existing skill and enthusiasm in the target-user scientific community.

7.2 Structured Social Events for Reducing Technological Uncertainty

In Section 6.1, I developed the idea that taking action can be associated with a great deal of uncertainty around what will be necessary in order to explore interest in a particular new tool or skill. Statements like, "I could do this all [day/week/month], I could let it take up that time, but I'll just see how far I get in [a more limited span] and that's that" are therefore not uncommon. However, the ability to stick to these self-imposed limits varies from person to person. A workshop event provides opportunity by way of its structure for something which there is already enough interest. Events provide structure in terms of time limitations as well as in terms of curriculum and helpers. The *existence* of this structure is an important benefit of the event: it makes the investment and leap of acting on an intention to implement some change more accessible. In other words, time limits and temporary "software-advisor" resources help build **opportunity** and concentrate **momentum**.

The RegionalNowcast-Model team had adopted not only specific technologies (iPython Notebook) but was also deliberately adapting some of the social protocols (such as feedback on code, as described, in Section 5.3). The PI of this group has been instrumental in starting and maintaining an annual SWC event for several years serving earth and ocean scientists[5]. Additionally, during the course of observations conducted in this study, this team also hosted a day-long "software sprint" event, which was preceded by an excited email to the attendees (which included both typical members, and a few new people who had not worked with this stack before), excerpted below:

> Rules: Maintain a respectful environment. Learning is a big part, if its most of what you accomplish in the day, that's just fine.

[5]I had made contact with this team in the context of an instructor training event that some of the group members had attended

> Rule of two feet: If its not working for you, if you don't feel you are learning or contributing, you should ask to work on something else, or you should move on.
> My Goals: team building, have fun, energize the stakeholders-request part of the project, everyone learn and if we get some advances in the code, that would be good too.
> Teams: we have google doc that I will send you all access to. It lists the projects. Based on these projects, we ask you to sit with a partner that is working on a synergistic project. As you work on projects, add notes, questions, comments to the google doc – choose a color!
> suggestions welcome, I haven't done this before! – RegionalNowcast-Model team PI

Like the joint visualization meeting hosted by a post-doc in the BioGeoChem-Model team (described in Sectio 6.3), the explicit goals of the RegionalNowcast-Model sprint were first and foremost interpersonal and awareness-building. In both cases, the intentions (to build awareness and to try out the tool-stack) were articulated informally and over email. As noted in Chapter 6, the uncertainty associated with "taking the plunge." The shared document mentioned in the email above hints at the way that this group uses persistent digital artifacts to maintain momentum and encourage participation. The PI of RegionalNowcast-Model group is updating the TODOs in the current shared document as it is projected during a weekly code-centered meeting: "it's great fun to see what everybody is doing! Very energizing!" Besides enthusiasm about code work, and her willingness to create a space where taking time to set up an effective working environment is okay (see email above), at least as important is her relentless closeness to the code work.

The PI of the CustomInstrument-Lab is a lot less engaged with the code work, partly because the members of the group that do code work have others from whom to seek help with code and they purposely try to use her time chiefly on "the science part." At one point she misunderstands a statement in a meeting and responds with a sense of urgency, "I

thought that was in there!" (regarding a piece of vital information). The three people most involved immediately jump in to reassure that the statement that precipitated the reaction is not nearly so dire (the data exists, but requires a join - it is both technically possible and practically plausible). When this PI attempts to "turn up the heat" and create some additional momentum and enthusiastic kind of pressure for creating a data visualization, this request is not armed with an accurate estimation of difficulty and does not carry as much weight. Code in such an environment settles almost immediately into invisibility when it works, and draws panic and dissatisfaction when it does not. This harsh judgment is omnidirectional: from the oceanographers to the computer science collaborators, but also from the oceanographers to themselves, often expressing frustration with tasks taking "too long" with "nothing to show for it." The different in recognition is not the enthusiasm of the PI - both are warm, genuine, encouraging, and use their financial resources to create room for innovation - but the *specificity* of the encouragement.

As I developed in the section on the collective imagination of the perfect world, discussing goals for code work that have elements of the presently-impossible is a common way to create direction in a highly uncertain environments. At some point, as the PI of the CustomInstrument-Lab group says that she has been promising that the visualization dashboard will be ready, because she "wanted to up the ante a bit," meanwhile George repeats - before and after her comment - that documentation is really important. After all, if the ante is up all the time, and code is in a state of near-constant emergency, not making time for documentation sets up larger problems later on: see *The Departure* story in Section 3.3 for more on this. If anything, this illustrates one of the primary challenges to code work in science: when a task is urgent, it is liable to be done in a way that incurs technical debt to be paid later; when it is merely important and not urgent, it goes by the wayside of other urgent and important tasks. Knowing when to uncompromisingly invest time into something that is not urgent but widely recognized as important takes self-discipline, social acceptance, and/or experience.

That said, it is not like the attitude of the RegionalNowcast-Model group makes them immune to rushing code and the inevitable resulting mess: Erin reflects on the work of an undergraduate summer research intern, whose project was "really cool, but really rushed... there is more clean-up we could have done... but I hope [the undergraduate intern] learned and had fun." Andrew, the in-house software advisor for the CustomInstrument-Lab, calls this "gradware:" published software "left to die," the physical manifestation of tensions between software production and academic success. This maps onto one of the problematic software production practices identified by Howison and Herbsleb, who write: "Software and algorithmic applications advance the theory but someone has to maintain and support the working software," which makes this "dual" practice of science and software not currently tenable [30]. In the CustomInstrument-Lab, the software advisor Andrew credits the PI of the team for allowing him, and others in her group (but outside the team studied) to focus on code work and therefore be able to invest time into maintenance.

PI of the RegionalNowcast-Model team makes an effort to bring achievements to the surface and to praise them, through both verbal acknowledgement and recognition, and through the use of a shared digital artifact, the "whiteboard." This is a GoogleDocs drawing, which is filled with dense areas of text in a mutli-column layout. Before their weekly meeting - a meeting that is specific to this particular project, and where specific code issues are raised, not only higher-level research planning and community-and-stakeholder-management issues - the PI (as well as other group members which I've shadowed prior to observing the meeting) check the board, and make any updates to reflect their progress. Everyone has a color that is associated with them; at one point, Ed the software advisor had made many changes and additions, and his color seemed to take up a lot of area, and this was recognized verbally in the beginning of the meeting. The meeting hinges on going through this document, acknowledging all the work that has been done, and resolving any outstanding issues. For each group member, the dynamic is a bit different; for example, one of the post-docs, Lindsay, spends most of her time grappling with FORTRAN code and adding parameters to the model, so her updates are sometimes as short as, "yep, still working on [mystery issue]," which elicits

either a friendly encouragement or some guidance on what could possibly be going on based on prior experience from either the PI or the software advisor.

Other group members, including junior graduate student Angela and post-doctoral research fellow Erin, often include links (in their individual colors, of course) to iPython Notebooks with their most recent scientific explorations of data. During the meeting, the PI clicks on these, and scrolls through, as the graphics and associated code and comments are displayed on the projector. The researcher who produced the charts talks through them, and each chart gets at least half a minute (usually considerably more) of silent consideration from the room, and a handful of follow-up questions. If I were late to such a meeting, and they were displaying the charts of a researcher and discussing them, I could guess just by the axes, the coloring, and the layout whose chart this was; this is not because different researchers have such distinct artistic preferences, but because the questions that they are exploring map onto a specific, distinct, and identifiable set of visual cues (see the last subsection of Section 2.3, which discusses some of the visual cues.)

The digital whiteboard document therefore gets filled over the course of a few meetings with links, updates, open questions and so on. Aside from organizing by color, the document is also sectioned off into a handful (half a dozen) areas of work. These include things like "Josh's Paper on [subject]" and "Upgrade." (The latter is described in the case study, *The Upgrade*). When it is filled, they create a new document, carrying over any information that is still necessary and relevant. At one point, as the team meets without the PI, who is out of town but who has encouraged them to meet without her, Erin considers the dense document and asks: "do you guys want to start a new board? It's starting to get really full..." and the verdict comes to "let it get really full!" with a hint of satisfaction at looking at this artifact of productivity. In a meeting with students, who had only recently began to work with the codebase, jump-started their involvement in the sprint, introduced above. The PI of the RegionalNowcast-Model team tells one of the students: "when the [date] nowcast is up, then you will see your arrow there!" She is referring to a particular analysis that he was ale to add during the sprint, with a great deal of pair-programming-type help from the PI

and other members of the RegionalNowcast-Model team and after several hours of setting up.

It is the quality, rather than the quantity, of external feedback that is important. The RegionalNowcast-Model team, has created and maintains an online visualization dashboard which shows the results of their models forecasting a day in the future ("nowcast") for a particular region. The software advisor of the RegionalNowcast-Model group talks about another group and how they do analytics over their software ("they care *a lot*," he tells me, as a follow up to telling me about how in the RegionalNowcast-model team, though they have a website, they do not collect analytics.) Collecting data on the number of visits to their site or more specifically to the documentation/tutorial page they maintain as a service to the broader community, the PI reads aloud the occasional email she gets to the tune of "this is great! thank you for doing this! we aspire to do this, too!" The quality, not quantity of feedback matters; this is also the case among some of Trainer et al.'s motivations for extra work [62]. And when George and Andrew, from the CustomInstrument-Lab, are talking about the software they are building and making it available on GitHub (which it is), the conversation readily acknowledges that only a handful of people could be considered potential users. This also comes up in interactions among the members of the BioGeoChem-Model team who were routinely spending time with older projects from prior research collaborations that someone has emailed them about, in a classic example of a motivation for "extra work" [62].

Take-Away for Evaluation

Focusing on technological deliverables and goals creates pressure; focusing on time spent cultivating community or learning a new vocabulary or way of thinking creates momentum. Events like the workshops and sprints have a variety of benefits and purposes aside from learning the content of the curriculum, or in achieving certain tasks. Both regarding the department-wide SWC workshop and the sprint for the common RegionalNowcast-Model tasks, the PI stresses the importance of (1) building community and

(2) learning something. Then, as the team members attending the sprint get each successive task done, she is increasingly enthusiastic, having set her expectations to be strategically exceeded. In the words of an interviewee from Chapter 6, the things that look uncertain and unapproachable sometimes (more often than not) turn out to "not be nearly so bad" - in many cases, momentum is needed rather than pressure, because the pressure is already shared by a group working toward a common goal.

Of the post-SWC interviewees, most were no longer using the specific skills taught a few months later, and some had some inaccuracies or confusions in their understanding of the concepts. However, all cited being much more confident in their own ability to seek out the right resources as needed, or engage with local or on-line community resources. One of the major benefits of time-limited events, like workshops and casual seminars, is that they allow for building community resource momentum *as a foundation* to additional code work. Not emphasizing concrete deliberables in the RegionalNowcast-Model team did not deter the participants from finishing large programming challenges facing the group; on the contrary, they remained in the lab beyond the official end of the event with a great deal of excitement and dedication to the actual projects. Afterwards, they felt energized, accomplished, and exhausted the next day: the downside of the sprint is that it is not a sustainable strategy for daily work, but rather a boost when it is needed.

7.3 Success Measures to Include Social and Cognitive Outcomes

In this section, I describe an event affecting the CustomInstrument-Lab, during which a relatively minor data corruption issue catalyzed a transition of tooling for the group. In particular, the database back-end had been used for two different applications, a real-time analysis visualization and an archive data endpoint that could be forward-facing. The initial implementation for a multi-purpose endpoint was done by David, (an external software advisor who had left, see Section 3.3) who worked closely with two post-docs, Mallory and George, on this project (it is his departure form the collaboration which is described in Section 3.3). Over the course of a year of observations, the question of whether this endpoint

ought to be split, how, what the features are, and discussion of anticipating user needs of other scientists was a recurring theme. An additional element was that of *who* would "own" this, as it went from the purview of David to that of Andrew (an in-house software advisor) who, as I note in the end of this chapter, was moving toward separating the functionalities.

Technologically, the event that took place was a hard-write failure on one of over a dozen machines that had been hosting a novel database developed by the computer science part of the team, and used by the oceanography part of the team. The corruption occurred (coincidentally) shortly after David's departure, and took a short time to reflect on until Andrew explored the needs of the problem and the capabilities of the available solutions enough to come to some course of action. The database, BigDB[6] was stored on these machines, and the write failure happened during the update of the database management system (DBMS) it was built on top of. Because the data was stored in an order particular to the DBMS and not obvious upon breakdown, the single write failure meant that (1) overall the database had been "corrupted" and (2) although most records were still there, it was not possible to tell *which* records were gone. The decision was made to "wipe" the database, and "re-ingest" the data.

Mallory, a post-doc in the team, was most affected by the corruption. "In a sense nothing is lost, but nothing is in a format I could do anything with," she comments of the data that must be re-ingested, but which is difficult since David was the one who had done that work initially. The situation is exacerbated by her receiving notification about it while she is on vacation when she gets the email, on the one day she set aside to make some "really cool charts for the PI." When I check-in with her shortly after her return to work, she describes the event as a "major set-back," "devastating," plus feeling some panic because has "nothing to show" for the time she had spent already.

Leading up to the data corruption event, the team had been meeting only sporadically. Occasionally, both the computing and the oceanography "sides" of the collaboration would

[6]Not its actual name, but the important feature of it was that is was well-suited to address a large data set.

articulate concerns. At one point one of the computing informants talked to me the difficulty of supporting a service in a research environment. An oceanography informant articulated frustration with feeling like they (the oceanographers) provide feedback and opportunity for the (computing) students to publish "fancy," complex things, while the computing students don't do "the basic plumbing." Meanwhile, on the subject of this kind of "basic plumbing," the concern from the computing "side" is that individuals working in this team had "spent enormous about of good will, energy, unpaid time getting basic plumbing, but the oceanographers were unwilling to help us make anything with a technical value proposition." The problem was not the data corruption, but instead communication of expectations: "[the oceanographers] didn't know what they were getting into, and we did not communicate that well enough; maybe noticed a mis-match early on, but did not do anything about it."

Focusing only on the tensions paints an unfair and incomplete picture of a complex, multi-year interpersonal relationship that resulted in exciting software and research products. The post-docs Mallory and George articulate a lot of pride and excitement over the computing skills they have gained during the course of the collaboration. Additionally, an interview from 2014[7] highlights this as a major benefit from the collaboration over time: "code written by biologists cannot be as good and as efficient as code written by a data scientist. So we are collaborating [with the computer scientists] to optimize, to refactor the code that we wrote," rather than telling computer scientists "now code this" because they realized right away that was not possible. In addition to being able to write "better" code, the computer scientists "may know better:" for example, if "there is a new clustering algorithm that has been developed that you need to read literature to understand." The computing researchers also derived satisfaction and meaning from being to able to have an impact on ocean science; for example, a computing PhD student, Leonard, spoke very positively about his experience: "The collaboration allowed me to do what you don't get to do a lot in [CS] grad school:

[7]Predating my study and collected by an RA in C.P. Lee's group at UW.

solve others' computational problems. ...The best thing is that there is never a point where I question, 'why am I doing this?"'

The on-line visualization endpoint that the database was serving - in addition to Mallory's analyses - was particularly sensitive to downtime, not only because it impedes analysis work, but also because it affects ability of the PI to "show off" all this "awesome data." A part of "reliability" is being forthcoming: in the case of a predecessor to BigDB, in-house software advisor Andrew notes that when it was down, he would email the people maintaining it, but that seemed "backwards" relative to how a service "should" conduct itself.

One of the frustrations had been repeatedly articulated by the ocean researchers from the beginning was either the inability to control something that they understand (like in one case where a query seemed way too slow, even upon attempts to reverse engineer a mechanism that would explain it), or the lack of understanding or information. Both frustrations exacerbated by limited documentation, and limited availability of the particular individuals who were the only ones who could explain. Unrelated to the data corruption event, when the oceanographers were looking for possible alternative tools, they invested time to try things out, read documentation, and look at tutorials. An interview from the beginning of the collaboration[8] notes: "When we collaborate, with [the computing] people, how can I explain this in a way that they can understand what the problems are? It seems like an easy thing to do. I think I can speak computer / data scientist language a little bit more than I used to. Before, many times when I say I have issues clustering [the data points in a particular way in post-processing], [they say] 'that's easy, don't worry. There's a lot of tools.' Even now that we are working with them for the three years, they are helping us, but they are struggling to realize that this is a big problem. This is not just '**oh, there are tools, gonna do this in one week, you're gonna have your [solution].**"' In other words, the speaker (from oceanography side) feels that the severity of the challenge is not recognized, and treated instead as a simple problem that can be solved easily with an existing tool.

[8]The same one from C.P. Lee's group

The resulting ambitious promises are then felt as doubly dismissive of the severity of the computing challenge when the solutions do not pan out, similar to the reaction in Section 7.1.

Reflecting on the "feature creep" that ended up contributing to the brittleness that caused the corruption to be felt by the users, a computing informant comments that that the CS group had a "culture of not really buying into the ability to be a service in addition to doing CS research." Furthermore, if not for the feature creep (of having their own back-end, and hosting the data on local machines, rather than using a more reliable cloud service), the "original proposal" (middle-ware) was "still the best solution, it's just not what's there." The same informant also expressed concern over excessive "hand-holding" which had prevented the oceanographers from becoming more robust in their recovery from a relatively minor issue.

In the wake of the data corruption issue, which occurred toward the very end of my observations, rather than using BigDB and its predecessor in every situation where they are somewhat plausible, the ocean researchers are pursuing the possibility of using different tooling solutions for the different technical problems they see. As one of the computing informants notes, the approach of having a specialized tool for every case is an impractical ideal: "well, it would be wonderful to have the exact right custom solution for every problem! but that is not the reality." This kind of customization and maintenance places overwhelming demands on the limited time and energy resources of someone like Andrew, an in-house software advisor to deal with it. The oceanography post-docs Mallory and George are cognizant of Andrew's limited time and only come to him with the most dire issues: "we can't [meaning, should not] be walking down the hall asking Andrew to do things all the time!" despite the sense that "well, he can just do the thing in the best way!" This is similar to the RegionalNowcast-Model group, where Ed's part-time in-house software advisor status exerted self-imposed limits even more severely.

Take-Away for Evaluation

The sense of having lost time, gone backwards, or ended up back in the beginning are common ways to describe particularly unpleasant data loss scenarios, or situations where one "gives up" on "trying to make [something] work." Even if there is a recovery that works as well as can be expected, it makes visible the brittleness of the work. Although the ocean researchers stopped using the service supported by computing researchers, the collaboration with computing researchers helped the ocean researchers develop a wider tool-set, skill-set, and comfort-level with a variety of technologies. The computing researchers, too, were able to get valuable feedback and meaningful work over the course of the collaboration. Furthermore, even if a breakdown provides an opportunity of change in the shape of being able to start again seemingly from scratch, it is impossible that nothing has changed in the course of the attempt that did not pan out.

The kinds of change that happen en route to the perfect world are not reversible. The cognitive-resource differences can be seen as past SWC participants struggle, as instructors, to remember what helped them to "get it" and what made a complicated concept "click" unless they specifically take notes during the process with the intention of reminding their future self of how they approached those struggles. Additionally, while no individual design or implementation error in a proposed intervention is "proof that an academic group cannot provide any tech service," the pattern of necessary pivots and adjustments without a solid grounding in communication and constructive reflection can seem dismissive, by both parties.

The momentum (in terms of skills acquired by the ocean post-docs) and initial set-up (in terms of existing solutions that can be improved, rather than an unapproachable sense of having to build every thing from scratch) afforded by the collaboration with the computing groups has been instrumental in doing cutting-edge ocean research. The data was "exciting" and "awesome," the papers were "great," part of a "good discovery;" "these data are too much for one person, for one lab, so the goal is to put all this data publicly available." Through these statements, the group members of the CustomInstrument-Lab

(on both "sides") recognized and validated community engagement and skill acquisition as intrinsically valuable outcomes of attempting to implement a particular solution to a particular problem, even if the solution is not the best or the problem changes over time. The take-away here is to **set initial, ambitious goals, against which outcomes will be measured, without unintentionally privileging any one part (technological, social, or cognitive) of the working environment over the others.**

7.4 Summary of Recommendations

The purpose of this chapter was to demonstrate the utility of the proposed conceptual framework, particularly with regard to evaluation of interventions. In each of the sections, I describe a case study of a complex situation that involved some adoption/adaptation, but also some rejection of particular intervention components, or their critique. The take-aways for each of the case studies are included below:

- When building something integrative or holistic, consider ways that its components can be made available piecewise and engage with the existing skill and enthusiasm in the target-user scientific community.

- Focusing on technological deliverables and goals creates pressure; focusing on time spent cultivating community or learning a new vocabulary or way of thinking creates momentum.

- Set initial, ambitious goals, against which outcomes will be measured, without unintentionally privileging any one part (technological, social, or cognitive) of the working environment over the others.

Taking these suggestions as a whole, the implication of this work for design and practice is to **articulate goals in ways that can reward community and skill related improvements**, because there is relatively less uncertainty associated with these than with

increasingly large or ambitious programming projects. The take-aways are intended to be useful for anyone about to invest time and energy in a project that carries with it an inevitable amount of risk.

In the parts of this sections where I expressly highlight criticism of some software intervention, I hope not to disparage the intervention, but to expose that the vocabulary for criticism of something that is broken is considerably more detailed than that for the opposite. One of WW's indictments about the conduct of the OOI (introduced in Section 7.1) is that "Rather than publicly acknowledging this failure and co-opting the extensive data management expertise and facilities that already reside within the NSF Geoscience community to solve this problem, the OOI is instead funding [external] engineers working at [another university] to cobble together a system at the last minute in the hopes that nobody will notice that anything has gone wrong" [67]. It is not the failure of a project under fire here so much as the failure of reflecting and moving past that failure.

The obvious opposite of criticism of brokenness - "praise of things which are not broken" - is neither actionable or timely. Criticism follows a breakdown, but functioning infrastructure is invisible. My intention in this chapter was to highlight other timely opportunities for encouragement and positive feedback. The first recommendation suggests recognizing hacks and workarounds that require manual intervention as not only clever (these usually exploit existing channels in new ways, and therefore *are* quite clever) but also revealing important information about how the "non-hacky-way of doing things" might be improved in design. The second recommendation offers a concrete way to cope with uncertainty by creating community momentum (through group work) and opportunity (through time-constrained effort) - and not measuring the outcome beyond this momentum and opportunity as it affects the particular context. The third recommendation stresses the necessity of articulating, in group settings, the learnings and change that *has* occurred in situations where there might be a sense of unsuccessful adoption, or "going right back to where we started" for all those affected, despite their different valuations.

In Chapter 5, I outline some of the best practices and claim that what they offer, upon adoption, is not so much a guarantee of success but a *language*. As SWC interviewees, quoted in Chapter 6, state, language and vocabulary are a major learning goal from attending such short-term educational initiatives. All these recommendations attempt to formalize looking on the bright side of possible failure, which ought to be seen as inevitable as projects involve more people, longer time spans, and more ambitious cross-overs between disciplines. Encouraging the sharing of negative results and disappointing data are among the top priorities for discussion of "best practices" in open science, and so should be the development of concrete ways to deliver interpersonal rewards in the wake of code work not panning out as well as hoped or intended.

Chapter 8

CONCLUSION

In this dissertation, I develop a conceptual framework based on code work done by four oceanography teams. In Chapter 7, I make recommendations for software and educational interventions, informed by this new framework. These findings may also be applicable in other scientific contexts where people with different disciplinary backgrounds use different methods to contribute to a integrated understanding, where data and code sharing are central but not necessarily standardized or embedded in a common infrastructure, and where programming activities span a variety of needs and orders of complexity. This framework offers a way of looking at the different forms of code work in an interconnected manner. This permits recognition of, and reasoning about, the extent to which deliberate changes in the way people write code impacts larger projects that can impact in fundamental ways the means of knowing.

As part of the framework, I introduced several concepts and definitions for articulating the context of *code work* done by scientists. A *working environment* includes tools and practices spanning data cleaning tasks, analytic tasks, manuscript management, and any other form of code work in the broader sense. Furthermore, the working environment accommodates *technical*, *social*, as well as *cognitive* resources. It is subject to change, driven by *momentum* and *opportunity*, which might be precipitate from a variety of factors like a breakdown of existing process or from the exciting beginning of a new project. The catalyst for the *moment of flux* is not limited to problems in dire need of a solution; it can also arise from curiosity, interest, and a sense of awe at the possibilities offered by adopting a particular technology. I defined these concepts in Chapters 4 and 6, and illustrated them in Figures 1.1 and 6.1.

A singular focus on problem-solving may marginalize opportunities for innovation that could drive community engagement, and, therefore, momentum and adoption. The deliberate acts of change in the moment of flux arise from action which is built up from awareness of, and intention to adopt, any of a variety of possible tools and skills. Workshops and tutorials help to built this awareness, and therefore are instrumental in changing code work over time, even if not directly resulting in immediate or sustained adoption. Adoption of an intervention, either a software tool or an education workshop, does not require a specific idea of how to integrate the new tool or skill into practice; more commonly, change is undertaken as a way to work toward a *collective imagination of a perfect world* where more is possible. While the decision to adopt something *is* associated with the hope that it will address an outstanding issue in a concrete way, the intervention must go beyond addressing a "pain-point." Furthermore, *sustained* adoption is not a requirement for the tool or skill to have a broader impact on the *collective imagination of the perfect world* that directs ongoing changes to the working environment.

Whereas Chapter 4 maintained the stance of characterizing how oceanographers do code work, Chapter 5 built on this dynamic, adding "best practices" as one facet of imagined the perfect world, as it is informed software engineering practice. Programming *best practices* constitute some of the most common components of the collective imagination of the perfect world because they offer a means for making code more understandable, usable (and re-usable), and extensible in the future, and improve its access to a broader audience with less individualized guidance effort needed. This maps onto reproducibility and open science values, which are part of the zeitgeist informing the inclusion criterion for this interview and observation-based study. When it comes to pursuing better practices - by modifying the working environment in the direction of the *best practices* in the perfect world - there are many options for adoption of particular tools, all of which require adapting the tools to the particular context of a project, for example relative to existing dependencies and any associated requirements. There are even more options for formalising social protocols for

communication and feedback, such as pair programming, code review, or specific protocols for documentation, which were explored in Chapter 5.

The conceptual framework I propose, along with the stories which illustrate it, is based on over 300 hours of interviews and observations with oceanographers across 4 academic groups in the Pacific Northwest, as well as participants of short-term programming education initiatives aimed at scientists (such as Software Carpentry Workshops). In conducting this qualitative study, I bridge my computer science training with research methods of my committee and colleagues in Computer-Supported Cooperative Work and Science and Technology Studies. This study had a wide breadth, including several very different scientific teams in the sample, and spanned 18 months. As such, it provides a snapshot of practices, and would be complemented by more in-depth longer-term studies, and comparative studies in other domains or after some years elapse.

Chapter 7 walks through three case studies, applying the perspective that the proposed conceptual framework enables, each of which concludes with a take-away for evaluation. The most important take-away from this dissertation is that engaging with software interventions, such as programming workshops or particular tools, offers community and skill benefits whether it results in sustained adoption. As software projects in science become more ambitious, transformation requires a language that recognizes and rewards the collective pursuit of uncertain possible worlds.

BIBLIOGRAPHY

[1] Steve Adolph, Wendy Hall, and Philippe Kruchten. Using Grounded Theory to Study the Experience of Software Development. *Empirical Software Engineering*, 16(4):487–513, 2011.

[2] Robert J Anderson. Representations and Requirements: The Value of Ethnography in System Design. *Human-computer interaction*, 9(3):151–182, 1994.

[3] Matthew Arrott, Alan Chave, Ingolf Krueger, John Orcutt, Alex Talalayevsky, and Frank Vernon. The approach to cyberinfrastructure for the ocean observatories initiative. In *OCEANS 2007*, pages 1–6. IEEE, 2007.

[4] Karen S Baker and Cynthia L Chandler. Enabling long-term oceanographic research: Changing data practices, information management strategies and informatics. *Deep Sea Research Part II: Topical Studies in Oceanography*, 55(18):2132–2142, 2008.

[5] Roger Barga, Jared Jackson, Nelson Araujo, Dean Guo, Nitin Gautam, and Yogesh Simmhan. The Trident scientific workflow workbench. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 317–318. IEEE, 2008.

[6] Adam Barker and Jano Van Hemert. Scientific workflow: a survey and research directions. In *Parallel Processing and Applied Mathematics*, pages 746–753. Springer, 2007.

[7] Susan M Baxter, Steven W Day, Jacquelyn S Fetrow, and Stephanie J Reisinger. Scientific software development is not an oxymoron. *PLoS Computational Biology*, 2(9):e87, 2006.

[8] Alessio Bechini and Anna Vetrano. Management and storage of in situ oceanographic data: An ECM-based approach. *Information Systems*, 38(3):351–368, 2013.

[9] Matthew J Bietz and Charlotte P Lee. Adapting cyberinfrastructure to new science: tensions and strategies. In *Proceedings of the 2012 iConference*, pages 183–190. ACM, 2012.

[10] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. Don't touch my code!: examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 4–14. ACM, 2011.

[11] Christine L Borgman. *Big data, little data, no data: scholarship in the networked world*. MIT Press, 2015.

[12] Christine L Borgman, Jillian C Wallis, and Matthew S Mayernik. Who's got the data? Interdependencies in Science and Technology Collaborations. *Computer Supported Cooperative Work (CSCW)*, 21(6):485–523, 2012.

[13] Ruy Cervantes, Bonnie Nardi, Kathy Graham, and Allan Kuchinsky. A new paradigm to support experience-sharing practices in experimental science and engineering.

[14] Nan-Chen Chen, Sarah S Poon, Lavanya Ramakrishnan, and Cecilia R Aragon. Considering time in designing large-scale systems for scientific computing. *arXiv preprint arXiv:1510.05069*, 2015.

[15] Cabell S Davis, Scott M Gallager, Martin Marra, and W Kenneth Stewart. Rapid visualization of plankton abundance and taxonomic composition using the Video Plankton Recorder. *Deep Sea Research Part II: Topical Studies in Oceanography*, 43(7):1947–1970, 1996.

[16] David Donoho. 50 years of data science. 2015.

[17] Steve M Easterbrook and Timothy C Johns. Engineering the software for understanding climate change. *Computing in Science & Engineering*, 11(6):64–74, 2009.

[18] Paul N Edwards. *A Vast Machine: Computer models, climate data, and the politics of global warming*. Mit Press, 2010.

[19] Neela Enke, Anne Thessen, Kerstin Bach, Jörg Bendix, Bernhard Seeger, and Birgit Gemeinholzer. The user's view on biodiversity data sharing: Investigating facts of acceptance and requirements to realize a sustainable use of research data. *Ecological Informatics*, 11:25–33, 2012.

[20] Michael Franklin, Alon Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record*, 34(4):27–33, 2005.

[21] Batya Friedman. Value-sensitive design. *interactions*, 3(6):16–23, 1996.

[22] John K Gilbert. Visualization: A metacognitive skill in science and science education. In *Visualization in science education*, pages 9–27. Springer, 2005.

[23] Keith Grochow. COVE: a visual environment for multidisciplinary science collaboration. In *Proceedings of the ACM 2009 international conference on Supporting group work*, pages 377–378. ACM, 2009.

[24] Keith Grochow, Bill Howe, Mark Stoermer, Roger Barga, and Ed Lazowska. Client+Cloud: evaluating seamless architectures for visual data analytics in the ocean sciences. In *Scientific and Statistical Database Management*, pages 114–131. Springer, 2010.

[25] P Guo. Data science workflow: overview and challenges. *blog@ CACM, Communications of the ACM*, 2013.

[26] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. How do scientists develop and use scientific software? In *Proceedings of the 2009 ICSE workshop on Software Engineering for Computational Science and Engineering*, pages 1–8. IEEE Computer Society, 2009.

[27] Dustin Heaton and Jeffrey C Carver. Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology*, 67:207–219, 2015.

[28] Kathryn Henderson. Flexible sketches and inflexible data bases: Visual communication, conscription devices, and boundary objects in design engineering. *Science, technology & human values*, 16(4):448–473, 1991.

[29] Bill Howe, Peter Lawson, Renee Bellinger, Erik Anderson, Emanuele Santos, Juliana Freire, Carlos Scheidegger, António Baptista, and Cláudio Silva. End-to-end eScience: Integrating workflow, query, visualization, and provenance at an ocean observatory. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 127–134. IEEE, 2008.

[30] James Howison and James D Herbsleb. Scientific software production: incentives and collaboration. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 513–522. ACM, 2011.

[31] Steven J Jackson and Sarah Barbrow. Infrastructure and vocation: field, calling and computation in ecology. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 2873–2882. ACM, 2013.

[32] Diane Kelly. Scientific software development viewed as knowledge acquisition: Towards understanding the development of risk-averse scientific software. *Journal of Systems and Software*, 109:50–61, 2015.

[33] Andrew J Ko, Brad Myers, Duen Horng Chau, et al. A linguistic analysis of how people describe software problems. In *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 127–134. IEEE, 2006.

[34] Andrew J Ko, Brad A Myers, and Htet Htet Aung. Six learning barriers in end-user programming systems. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 199–206. IEEE, 2004.

[35] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. Technical debt: from metaphor to theory and practice. *Ieee software*, (6):18–21, 2012.

[36] Robert Kunzig. *Mapping the Deep: the extraordinary story of ocean science*. WW Norton & Company, 2000.

[37] Bruno Latour and Steve Woolgar. *Laboratory Life: The construction of scientific facts*. Princeton University Press, 2013.

[38] James Leach, Dawn Nafus, and Bernhard Krieger. Freedom imagined: morality and aesthetics in open source software design. *Ethnos*, 74(1):51–71, 2009.

[39] Charlotte P Lee. Boundary negotiating artifacts: Unbinding the routine of boundary objects and embracing chaos in collaborative work. *Computer Supported Cooperative Work (CSCW)*, 16(3):307–339, 2007.

[40] Charlotte P Lee, Matthew J Bietz, Katie Derthick, and Drew Paine. A sociotechnical exploration of infrastructural middleware development. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1347–1350. ACM, 2012.

[41] Charlotte P Lee, Paul Dourish, and Gloria Mark. The human infrastructure of cyberinfrastructure. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 483–492. ACM, 2006.

[42] Jie Li, Zhao-Peng Meng, and Kang Zhang. Visualization of oceanographic applications using a common data model. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 933–938. ACM, 2014.

[43] Matthew B Miles, A Michael Huberman, and Johnny Saldaña. *Qualitative data analysis: A methods sourcebook*. SAGE Publications, Incorporated, 2013.

[44] KAS Mislan, Jeffrey M Heer, and Ethan P White. Elevating the status of code in ecology. *Trends in ecology & evolution*, 31(1):4–7, 2016.

[45] Drew Paine and Charlotte P Lee. Producing Data, Producing Software: Developing a Radio Astronomy Research Infrastructure. In *e-Science (e-Science), 2014 IEEE 10th International Conference on*, volume 1, pages 231–238. IEEE, 2014.

[46] Marc Palyart, David Lugato, Ileana Ober, and Jean-Michel Bruel. Improving scalability and maintenance of software for high-performance scientific computing by combining MDE and frameworks. In *Model Driven Engineering Languages and Systems*, pages 213–227. Springer, 2011.

[47] Kayur Patel, James Fogarty, James A Landay, and Beverly L Harrison. Examining Difficulties Software Developers Encounter in the Adoption of Statistical Machine Learning. In *AAAI*, pages 1563–1566, 2008.

[48] Pierrick Penven, Patrick Marchesiello, Laurent Debreu, and Jérome Lefevre. Software tools for pre-and post-processing of oceanic regional simulations. *Environmental Modelling & Software*, 23(5):660–662, 2008.

[49] F Perez. The IPython notebook: a historical retrospective. *Jan*, 2012.

[50] Kavita Philip, Medha Umarji, Megha Agarwala, Susan Elliott Sim, Rosalva Gallardo-Valencia, Cristina V Lopes, and Sukanya Ratanotayanon. Software reuse through methodical component reuse and amethodical snippet remixing. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1361–1370. ACM, 2012.

[51] Benjamin Ragan-Kelley, William Anton Walters, Daniel McDonald, Justin Riley, Brian E Granger, Antonio Gonzalez, Rob Knight, Fernando Perez, and J Gregory Caporaso. Collaborative cloud-enabled tools allow rapid, reproducible biological insights. *The ISME journal*, 7(3):461–464, 2013.

[52] OJ Reichman, Matthew B Jones, and Mark P Schildhauer. Challenges and opportunities of open data in ecology. *Science*, 331(6018), 2011.

[53] Russ Rew and Glenn Davis. NetCDF: an interface for scientific data access. *Computer Graphics and Applications, IEEE*, 10(4):76–82, 1990.

[54] David Ribes and Thomas A Finholt. The long now of technology infrastructure: articulating tensions in development. *Journal of the Association for Information Systems*, 10(5):5, 2009.

[55] Steve Sawyer. Data wealth, data poverty, science and cyberinfrastructure 1. *Prometheus*, 26(4):355–371, 2008.

[56] Reiner Schlitzer. Interactive analysis and visualization of geoscience data with Ocean Data View. *Computers & geosciences*, 28(10):1211–1218, 2002.

[57] Richard Sennett. *The craftsman*. Yale University Press, 2008.

[58] Magnus Thorstein Sletholt, Jo Hannay, Dietmar Pfahl, Hans Christian Benestad, and Hans Petter Langtangen. A literature review of agile practices and their effects in scientific software development. In *Proceedings of the 4th international workshop on software engineering for computational science and engineering*, pages 1–9. ACM, 2011.

[59] Susan Leigh Star. The ethnography of infrastructure. *American behavioral scientist*, 43(3):377–391, 1999.

[60] Stephanie B Steinhardt and Steven J Jackson. Anticipation work: Cultivating vision in collective practice. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 443–453. ACM, 2015.

[61] Victoria Stodden and Sheila Miguez. Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. *Available at SSRN 2322276*, 2013.

[62] Erik H Trainer, Chalalai Chaihirunkarn, Arun Kalyanasundaram, and James D Herbsleb. 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*, pages 417–430. ACM, 2015.

[63] Sharon Traweek. *Beamtimes and Lifetimes: The World of High Energy Physicists*. Harvard University Press, 2009.

[64] Duane Truex, Richard Baskerville, and Julie Travis. Amethodical systems development: the deferred meaning of systems development methods. *Accounting, management and information technologies*, 10(1):53–79, 2000.

[65] Janet Vertesi. "Seeing like a Rover": Images in Interaction on the Mars Exploration Rover Mission. 2009.

[66] Kirsten N. Whitley and Alan F Blackwell. Visual programming in the wild: A survey of LabVIEW programmers. *Journal of Visual Languages & Computing*, 12(4):435–472, 2001.

[67] William Wilcock. The Shambolic State of the Ocean Observatories Initiative. `http://faculty.washington.edu/wilcock/?p=240`, 2014. Accessed: 2016-01-29.

[68] Greg Wilson. Software Carpentry. *Computing in Science & Engineering*, 8:66, 2006.

[69] Greg Wilson. Software Carpentry: Lessons Learned. *F1000Research*, 3:62–62, 2013.

[70] Greg Wilson, DA Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven HD Haddock, Katy Huff, Ian M Mitchell, Mark D Plumbley, et al. Best practices for scientific computing. *PLoS biology*, 12(1):e1001745, 2014.

[71] Langdon Winner. *The whale and the reactor: A search for limits in an age of high technology*. University of Chicago Press, 2010.

[72] Tamiji Yamamoto. Small-scale variations in phytoplankton standing stock and productivity across the oceanic fronts in the southern ocean. *Memoirs of National Institute of Polar Research. Special issue*, 40:25–41, 1986.

[73] Ron B Yeh and Scott Klemmer. Field Notes on Field Notes: Informing Technology Support for Biologists. 2004.

[74] Alyson L Young and Wayne G Lutters. (Re) defining Land Change Science through Synthetic Research Practices. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 431–442. ACM, 2015.

[75] Ann Zimmerman. Not by metadata alone: the use of diverse forms of knowledge to locate data for reuse. *International Journal on Digital Libraries*, 7(1-2):5–16, 2007.

Appendix A
INTERVIEW PROTOCOLS

When interviewing participants in the SWC post-interviews group (see Figure 3.1), I asked the following questions:

- How did you decide to go to SWC? What was one surprising thing you learned?
- Tell me about something that you learned that you incorporated into your [data science workflow / analysis pipeline - the phrasing reflected anything that the participant can used, defaulting to "data science workflow" but relying on follow-up questions to broaden the scope to instances of "casual" code work that may have been initially omitted.]
- Followup questions about lab mates, or if not mentioned: did other people in your lab or group go? Are there people you work closely with, or of whom you ask questions when you need help? How would you characterize your role, and the roles of others in your group?
- List top 5 data science tools or resources.
- If there was a perfect event, course, or resource for you to learn what you're most interesting in learning about doing data science, what would that be like?
- *For follow-up interviews:* What's something interesting that you learned to do or make recently, in terms of collecting cleaning analyzing or presenting your data or results?

As part of the observation protocol, I asked questions about recent times that people had: created a chart or visualization; cleaned or processed data; analyzed data; worked on a paper, presentation, poster, talk, or other communication-oriented artifact; programming,

scripting, coding, tuning model parameters, and so on; explaining code or data to others. Questions about "what are you currently working on" included follow-ups such as:

- What's the current thing you're working on? How long have you been working on it? What are you stuck on?
- What other people are you closely working with? Who else is involved in this project?
- What are you currently struggling with? Describe the current challenge.
- What is something you recently decided to stop doing, or start doing differently?
- How do you decide what to work on next?
- How do you make your code or software useful?
- *In the beginning of the day:* What are the things you plan or hope to do today?
- *At the end of the day:* How has today gone, relative to your expectations and hopes in the beginning of the day?

In addition to interviews I conducted with past SWC attendees and oceanographers during the course of year-long observation, the study analysis has included interviews conducted by UW colleagues in 2013 and 2014 with oceanographers, which asked the participants: "what projects are you currently working on?" and related followups; "what are your funding sources?"; "what software do you use / for what / how did that come about?"