Requirements Engineering in Building Climate Science Software

by

Archer L. Batcheller

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Information)
in The University of Michigan
2011

Soli Deo gloria.

Acknowledgements

From birth, and to the present, my family has helped provide an environment were I could flourish. I am indebted to my parents, Gordon and Mary Batcheller, who taught me how to work and learn, and have unwaveringly encouraged me in the pursuit of my interests. I am also most appreciative of my wife Kimberly, and daughters Lily, Noelle, and Clara. They have been patient with me while I have been submerged in the dissertation work, and brought me great joy whenever I surfaced. Kimberly's support has been invaluable, from verbal affirmations to feedback about my ideas. I treasure her friendship, and am glad to live life with her for the rest of our days. She readily fulfills the scripture: "An excellent wife is the crown of her husband," and she is "far more precious than jewels" (Proverbs 12:4, 31:10, English Standard Version).

I have enjoyed and benefitted from the positive culture in the School of Information at the University of Michigan that provides a great environment for research. I am thankful for friendly and encouraging officemates, who provided camaraderie as we worked on our dissertations. This is especially true for Lian Jian, John Lin, Brian Hilligoss, and Jude Yew. I am grateful for encouragement and prayers from John Lin and David Lemmerhirt. I would be remiss to fail to mention the excellent support staff at the School of Information. I have benefited from the aid of many, but am particularly appreciative of the expertise and smiles of Christine Eccleston and Sue Schuon as they handle their respective jobs with excellence.

My committee members invested their time reading drafts and crafting feedback for me. I appreciate their honest and reflective comments that have both focused me on details and helped me look at the big picture. Thank you, Paul Edwards, Ann Zimmerman, Christiane Jablonowski, and Steve Easterbrook. I have had helpful conversations with each committee member individually, and each has introduced me to people that yielded further fruitful dialog. I am especially indebted to my chair and advisor, Paul Edwards. As we have worked together I have becoming increasingly impressed with his adept mentoring and encouragement. In addition to his practical advice, he would weigh my personality, skills, and the challenges of the dissertation to provide balanced feedback that both challenged and encouraged me in my work.

I am the grateful beneficiary of National Science Foundation Award #0827316, which funded my appointment as a research assistant on the project "Monitoring, Modeling, and Memory: Dynamics of Data and Knowledge in Scientific Cyberinfrastructure." My participation in this project introduced me to the communities I studied in this dissertation and catalyzed my interest in teams building climate science software. I was glad to be able to simultaneously investigate the research questions in my dissertation and in the NSF project. Furthermore, I benefited immensely from the interaction and insights of my team members: Paul Edwards, Steve Jackson, Ayşe Büyüktür, Geof Bowker, Leigh Star, David Ribes, Chris Borgman, Jillian Wallis, and Matt Mayernik.

The engineers, scientists, and managers that I interviewed were generous with their time, and thoughtful in their comments. I am most appreciative of their willingness to interrupt their schedules to meet with me, and it is their perspectives in aggregate that made the insights in this thesis possible. I hope that my findings may prove helpful as they seek to build effective systems and support scientific work.

Traditionally, I pray desperately to God when faced with a difficult set of deadlines and work, and then take credit for my own efficiency and focus when recounting the events later. God has poured out his mercy and help once again in answer to my prayers and those of others praying on my behalf. In the midst of a new baby and imminent move to start a new job, He has helped me to focus, think, and write. In the face of looming milestones, He has granted me conceptual insights that instantly clarified a chapter's worth of content. God's word has encouraged and challenged me to work diligently and joyfully, and to be bold in taking on a project that could seem like a fearsome giant. Rather than take credit for this, I want to simply acknowledge God's goodness and graciousness to me. To God alone be glory.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AGU – American Geophysical Union
AMIP – Atmospheric Model Intercomparison Project
AR – Assessment Report, published by the IPCC
BEI – Battlespace Environments Institute, a DOD institute
CCSM – Community Climate System Model
CDAT – Climate Data Analysis Tools
CES – climate science Computational End Station
CESM – Community Earth System Model
CF – Climate and Forecast (metadata convention)
CGD – Climate and Global Dynamics, an NCAR division
CMIP – Coupled Model Intercomparison Project
CMIWG – Common Modeling Infrastructure Working Group
CRB – Change Review Board, an ESMF board
CSM – Climate System Model
CVS – Concurrent Versioning System
DOD – Department of Defense
DOE – Department of Energy
ESG – Earth System Grid
ESG-CET – ESG - Center for Enabling Technologies
ESMF – Earth System Modeling Framework
ESRL – Earth Systems Research Laboratory, a NOAA laboratory in Boulder, CO
FMS – Flexible Modeling System, used at GFDL
FTP – File Transfer Protocol
GEOS-5 – Goddard Earth Observing System, version 5
GFDL – Geophysical Fluid Dynamics Laboratory, a NOAA laboratory in Princeton, NJ
GO-ESSP – Global Organization for Earth System Science Portals
GrADS – Grid Analysis and Display System
GRIB – GRIdded Binary (data format)
GSFC – Goddard Space Flight Center, a NASA center in Greenbelt, MD
HDF – Hierarchical Data Format
ICSE – International Conference on Software Engineering
IDL – Interactive Data Language
IPCC – Intergovernmental Panel on Climate Change
LLNL – Lawrence Livermore National Laboratory

MAP – Model Analysis and Prediction, a NASA program
MAPL – Modeling, Analysis and Prediction Program Layer
MCT – Model Coupling Toolkit
MMM – Monitoring, Modeling, and Memory (research project)
MPI – Message Passing Interface
NASA – National Aeronautics and Space Administration
NCAR – National Center for Atmospheric Research, an NSF center in Boulder, CO
NCL – NCAR Command Language
NESII – NOAA Environmental Software Infrastructure and Interoperability
NetCDF – Network Common Data Form
NOAA – National Oceanic and Atmospheric Administration
NSF – National Science Foundation
NUOPC – National Unified Operational Prediction Capability
OPeNDAP - Open-source Project for a Network Data Access Protocol
ORNL – Oak Ridge National Laboratory
PCMDI – Program for Climate Model Diagnosis and Interpretation (at LLNL)
PI – Principal Investigator
PRISM – PRogramme for Integrated Earth System Modeling
SciDAC – Scientific Discovery through Advanced Computing, a DOE program
UCAR – University Consortium for Atmospheric Research
WRF – Weather Research and Forecasting (model)

Glossary

Component – a portion of a model that represents a subset of the larger system
Model – a representation of a physical system, which simulates the behavior of the world
Refactor – changing software code to be clearer, more efficient, or use a better design
Requirements – *verifiable* properties that are to be fulfilled by the design
  Business requirements – project goals, objectives, and desired outcomes
  User requirements – user goals and tasks to perform with the system
  Process requirements – specify how system development should happen
  Product requirements – specify the properties of the system to be developed
  Functional requirements – what the system should do
  Nonfunctional requirements – how the system should do it
Stakeholders – actors who influence requirements or who are impacted by the system

Abstract

Software has an important role in supporting scientific work. This dissertation studies teams that build scientific software, focusing on the way that they determine what the software should do. These requirements engineering processes are investigated through three case studies of climate science software projects. The Earth System Modeling Framework assists modeling applications, the Earth System Grid distributes data via a web portal, and the NCAR (National Center for Atmospheric Research) Command Language is used to convert, analyze and visualize data. Document analysis, observation, and interviews were used to investigate the requirements-related work.

The first research question is about how and why stakeholders engage in a project, and what they do for the project. Two key findings arise. First, user counts are a vital measure of project success, which makes adoption important and makes counting tricky and political. Second, despite the importance of quantities of users, a few particular "power users" develop a relationship with the software developers and play a special role in providing feedback to the software team and integrating the system into user practice.

The second research question focuses on how project objectives are articulated and how they are put into practice. The team seeks to both build a software system according to product requirements but also to conduct their work according to process requirements such as user support. Support provides essential communication between users and developers that assists with refining and identifying requirements for the software. It also helps users to learn and apply the software to their real needs. User support is a vital activity for scientific software teams aspiring to create infrastructure.

The third research question is about how change in scientific practice and knowledge leads to changes in the software, and vice versa. The "thickness" of a layer of

software infrastructure impacts whether the software team or users have control and responsibility for making changes in response to new scientific ideas. Thick infrastructure provides more functionality for users, but gives them less control of it. The stability of infrastructure trades off against the responsiveness that the infrastructure can have to user needs.

Chapter 1

Introduction

Historically, scientific and technical advances have often gone hand in hand. Technological improvements have allowed incrementally better scientific work. Simultaneously, the requirements for scientific investigation have demanded better technologies. Recently, computing advances have spurred on science in tremendous ways. Just as computer capacities advance exponentially with Moore's law, computing contributions to science have also exploded. Jim Gray and others declared these changes a qualitative shift, introducing an entirely new paradigm of scientific work (Hey et al. 2009). Thus the study of "e-science," or the production of "scientific cyberinfrastructure," has gained prominence as a key to maximizing research productivity.

However, creating software tools for scientists is not a simple process. Commercial software companies produce some scientific tools, but often the development of complex scientific software is done by research teams with tight budgets. Generating effective systems is tricky, involving the alignment and consideration of standards, work practices, norms, institutions, and technology. Furthermore, all these advances in computing require real engineering work. At the same time, science and the requirements for scientific software are changing as research yields new findings. Software engineering and design methodologies critically impact the manner and types of software produced. Engineering in this context differs from engineering in industry in dimensions such as the way code is used and requirements are determined.

Some scientists develop their own software while others delegate that task to software developers or computer science collaborators. Here I focus on projects that have teams of dedicated software engineers hired to produce software for scientists. This is an important model of development to consider because it represents groups that have enough resources and enough potential scientist "users" to warrant a dedicated team. Furthermore, the primary goal of such teams is to produce good software for their users, rather than doing research in computer science or some other domain. But how well does the software engineering training of these developers work in the science context? In particular, how do software engineers relate to scientists and come to understand the requirements for the software they are producing? This process of requirements engineering within teams producing scientific cyberinfrastructure is the focus of this dissertation. This work is critically important for understanding and improving the way that teams go about designing software for scientists, which can help to more efficiently produce effective software. Particular insights about requirements engineering are more clear in different contexts, so I believe that this work contributes to a richer understanding of requirements engineering by broadening the types of software development environments considered in the literature. My hope is that this work will also have direct benefit for teams working on scientific software.

Introductory computer science students are taught that despite steadily better computer hardware, algorithmic gains and losses can dwarf the effects of faster processors. If we sort one million items into an order, a poor algorithm requires on the order of 10^{12} operations. A better algorithm requires on the order of 10^{7} operations, some 100,000 times better. When processing speeds were doubling every 18 months, it would still require 25 years of such gains to make up for using the slow algorithm. Algorithmic innovation and choices can make a fantastic difference. Yet the implications of good design processes and proper requirements engineering are even more drastic.

A program may run slowly enough to make some operations impossible. However, a system that is poorly designed can go altogether unused. Some projects are never completed; others are completed but do the wrong thing; still others are completed and do the right thing but are never used. It is critical to have design methodologies for scientific projects that carry a team successfully through all stages of the process and guide communication between engineers and scientists to yield a product that satisfies their needs. That is, software teams need to pursue good processes, not just good software. Requirements engineering is crucial to success for scientific software, and is thus the focus of this dissertation.

1.1 Reading guide and summary of key findings

The remainder of this chapter is a general literature review, with more specific literature introduced as needed within the main chapters. The introductory literature is organized as follows: The section on "Software tools for scientists" discusses the movement to intentionally develop software to support scientific work, sometimes framed as "cyberinfrastructure." It concludes by highlighting the paucity of methods for designing scientific software, and particularly the challenge in identifying requirements in large, distributed, and dynamic projects. This first section focuses on science and technology studies, computer supported cooperative work, and information science literature. The next section, "Particularities of scientific software," provides important background about the qualities of scientific software and software engineering practices for the sciences. This section is intended to provide a contrast between software engineering for industry and software engineering for science to aid in understanding requirements engineering for scientific software. The literature reviewed here is primarily from the field of software engineering. The section on "Requirements for scientific software" focuses on findings from requirements engineering literature that are relevant to scientific software. The last three sections continue to discuss requirements engineering, with each corresponding to the ideas in the core sections of this thesis, Chapters 3-5.

Chapter 2 addresses my methodological approach, and provides details about how I conducted this research study. It also explains why I have chosen the three software projects that serve as my case studies, and gives some general background about what each software system does, the history of the team, and a little bit about the organization of each team. As with the literature, more specific details about the case studies will be introduced in the chapters as they are needed.

Chapter 3 is the first of three main chapters that both introduce data and develop theory. In Chapter 3, I focus on mapping out the roles of the various stakeholders that engaged in the software projects in my case studies. The first part of this chapter details how and why stakeholders engage in the project, and what they do for the project. This yields a series of smaller observations, such as the important role that project managers play in stabilizing the project from the perspective of other stakeholders. However, two key findings fall out of this investigation of stakeholders. First, user counts are a vital measure of project success, which makes adoption important and makes counting tricky and somewhat political. Second, despite the importance of quantities of users, a few particular "power users" develop a relationship with the software developers and play a special and irreplaceable role in helping with the development of the system and its integration into user practice.

Chapter 4 investigates the way that projects state their objectives. I find a formal narrative in project documents that is oriented around software functionality, while the informal narratives of team members emphasize team processes. This serves to reveal the different types of work done by the software team, including both the development of a software system according to product requirements but also the conduct of user support and non-programming activities according to (often unstated) process requirements. The latter half of Chapter 4 focuses on the value and work accomplished through one of the main process requirements: user support. Support provides essential exchange between users and developers that assist with refining and identifying requirements for the software. It also helps users to learn the software in a specific way that applies it to their real needs. I argue that user support is a vital activity for scientific software teams aspiring to create infrastructure, and support distinguishes such projects from scientific software teams that are more focused on research.

Chapter 5 is concerned with how change in scientific practice and knowledge leads to changes in the software, and vice versa. This issue is a central one for requirements engineering, as it focuses on how a team adjusts to the changing functions that its software must accomplish and how decisions about the software requirements impact the users. Scientific software aspires to improve and change scientific practice, and yet the software systems must also respond to meet the needs of ever-changing scientific practice. I first compare and contrast my simple model of software infrastructure to three significant models from the literature. Then I apply my model to the data, with two main findings. First, the "thickness" of the infrastructure layer impacts whether the software team or users have control and responsibility for making changes in response to new scientific ideas. Thick infrastructure provides a great deal of functionality for users to draw upon in doing their work, but it also means that they have less control over the software if they want to change an algorithm to try out a new idea. This leads to a second related point: there is a tradeoff between the robustness/stability of infrastructure and the flexibility/responsiveness that the infrastructure can have to user needs. I conclude by noting how this tension is one reason for layers of infrastructure that are increasingly tailored for the needs of a particular community, placing control and flexibility for some parts of the software nearer to end users.

Chapter 6 is the conclusion, which highlights the key theoretical contributions of these findings and suggests some implications both for policy and practice. I close with a few possible directions for future work.

1.2 Software tools for scientists

There has been much recent attention to building software tools for scientists. The Atkins et al. blue ribbon report to the National Science Foundation recognized a vital need for science to develop capacity to take advantage of computational advances (2003). The authors advocated development of an infrastructural layer – known as scientific cyberinfrastructure – that would support scientific work and further application development. For instance they suggested investing in networking, supercomputers, and framework software to support scientists who are developing software-based simulations. The scientists then no longer need to be concerned with the infrastructural layers that are mature and reliable to support their computational needs. It is important to note that this conceptualization focused on middleware, assuming that others will continue to produce innovative hardware at a lower layer and software applications at an upper layer.

In order to advance thinking on infrastructural support, the National Science Foundation (NSF) elaborated on the Atkins report through a series of domain-specific reports of cyberinfrastructure needs, and later by convening a workshop on the history of infrastructure. The NSF workshop on the History and Theory of Infrastructure led to the "Understanding Infrastructure" report, which provided a new definition of scientific cyberinfrastructure that broadened the scope of interest and deemphasized the middleware layer:

> "…cyberinfrastructure is the set of organizational practices, technical infrastructure and social norms that collectively provide for the smooth operation of scientific work at a distance." (Edwards et al. 2007)

For my purposes in this dissertation, I will adopt this definition with its broad scope for considering the software tools of scientists. Multiple parts of the scientific environment can be engineered, not just the technical. We can helpfully thus refer to cyberinfrastructure as a *sociotechnical system*, comprised of both technical and social components. Technical components include computer code, hardware, or standards, while social components include people, organizations, and social norms. Studies of traditional infrastructure like railroads and electricity find that accounts centered on technology or physical artifacts are inadequate. Instead, technology that supports an infrastructure must go through social negotiation and standardization. New business models and work practices must accompany new infrastructure. Supply and demand for an infrastructure grow together. For this reason, science and technology studies has approached infrastructure as a holistic, sociotechnical system. Star and Ruhleder find infrastructure to be fundamentally characterized by its visibility upon breakdown and broad usage within a certain community (Star and Ruhleder 1996). When working properly, infrastructure is ready-at-hand, reliable, and also invisible.

The "Understanding Infrastructure" report drew lessons from how previous infrastructures have grown to be a fundamental tool for work to suggest how scientific cyberinfrastructure may accomplish the same ends. Infrastructures are influenced by path dependencies, continuing down a route once some early decisions have been made. Infrastructures carry tremendous inertia making them difficult to shift (Hughes 1987). In light of these observations, the report recommends design approaches such as planning for flexibility and change, aligning incentive structures, and building in mechanisms to diagnose infrastructural dynamics.

The goal of building scientific cyberinfrastructure is to create a set of building blocks upon which scientists can draw while doing their work. Sometimes this is framed as building middleware that sits between hardware and the software and supports commonly needed functionality like sharing computing, storage, or data cataloging facilities. Other times it involves writing software frameworks that allow scientists to write some code that takes advantage of the functions of the software system. In practice, this also often means that cyberinfrastructure is *not* built by the domain scientists whom it supports. Instead, it may be developed by supporting engineers and software developers who have time, knowledge, and interest in that type of work. For instance, software engineers would develop a modeling framework, and domain scientists may then create the models that make use of it. In reality, these are complicated relationships with people who play multiple roles and have overlapping interests. However, in general the incentives and interests for scientists lead them to want to do science with supporting infrastructures that minimally disrupt this work.

While engineers may primarily handle cyberinfrastructure, this does not mean that scientists are absent from cyberinfrastructure design work. In fact, funding agencies have tended to place scientists centrally in cyberinfrastructure projects, diluting the projects' missions to both engineering and research work. This has led some cyberinfrastructure teams to partner with computer scientists to help develop new systems. Weedman and Lawrence both document some of the trade-offs that occur in this type of team structure (Weedman 1998; Lawrence 2006). Partnering with computer scientists may open up new possibilities of funding, and expertise in advanced system building. However, computer scientists are researchers too, and have their own research questions and goals for the project. Understandably, they are reluctant to invest significant time and energy into debugging and fine-tuning the software system, as career structures in computer science do not significantly reward development work beyond proof-of-concept prototyping (Ribes and Finholt 2007). Yet scientists who want to use the software need a reliable and usable, production-ready system. The results of Weedman and Lawrence's research warn cyberinfrastructure design teams to carefully consider the stakeholders participating in the project, including the software developers (Weedman 1998; Lawrence 2006).

The National Science Foundation's workshop on "Planning for Cyberinfrastructure Software" recognizes software as a critical part of infrastructure for scientists, and one that "should be the target of explicit design, construction, study, and evolution" (Blatecky and Messerschmitt 2005). The report acknowledges that cyberinfrastructure software development is distinct from research on cyberinfrastructure software, and that it needs to be deliberately and proactively supported, perhaps better than it had been to date. The attention to software design and development requires interaction and blurring between engineers and scientists. As they propose: "Just asking about requirements is a tiny part of the problem – computer science developers must learn to 'walk in the shoes' of the scientists."

In short, the report is hinting that developing software for scientists requires close collaboration between domain scientists and software engineers. At times, domain scientists may be deeply involved in the software design and development process, not merely as users of the software. But is this form of software development any different than what occurs in industry or open source? Good design practices for software cyberinfrastructure are important, both saving money and even making a qualitative difference about whether a project succeeds. How then may designing for scientists be special?

Software engineering is "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" (IEEE 1990). To study the full range of software engineering practices entailed in this definition, we need to consider the formal and informal, social and technical, from beginning to end. The recent cyberinfrastructure reports direct our attention particularly to the design aspects of cyberinfrastructure, where a few recent studies have begun to document the challenges incumbent in even determining cyberinfrastructure requirements, a prerequisite to making design choices.

For cyberinfrastructure development, it is important to focus on the "human infrastructure" around the technology as well (Lee et al. 2006). "Distributed teams" are a weak framing for situations with fuzzy group membership such that people are uncertain as to which “teams” they belong. Often fuzzy membership is further aggravated by collaboration technologies that do not support the same ambiguity of group membership and participation as face-to-face collaboration affords. Individuals have interests that do not fully align with the simple success of a technical system: “The cyberinfrastructure provides a means of producing and transforming local concerns – institutional prestige, academic power relations, organizational relationships, access to appropriate scientific data, access to subjects, and so on” (Lee et al. 2006). In planning cyberinfrastructure, designers need to be conscious of these mixed allegiances in which the primary requirements for some stakeholders may not be the material success of the system but rather the advancement of some other personal, institutional, or disciplinary objectives.

Ribes and Baker classify various ways that social scientists engage in cyberinfrastructure work (2007). One way they are included in the projects is as authorities to help with requirements analysis, particularly facilitating participatory design or user-centered design activities. Zimmerman and Nardi reflected on just such an experience of design activity, and found themselves frustrated by the inadequacy of current HCI methods (2006). This is a critical issue. They warn: “A major challenge for the design of cyberinfrastructures is to support diverse users at multiple sites under conditions of rapid change. There is significant risk that investments in cyberinfrastructures will not be optimized unless this challenge is met.” In sum, they found that user-centered design was simply inadequate for the complexity of a large cyberinfrastructure project like the National Ecological Observatory Network. That NSF-funded network involves multiple sites across the country, incorporates a diverse base of users spanning a whole discipline, and is envisioned as a long-term project.

The crux of the problem is that cyberinfrastructure is intended to transform the way that science is done, so designers are trying to hit a moving target that is not even yet in sight! Cyberinfrastructures “are unstable because the practice of the science is being redesigned as the cyberinfrastructure is being designed and deployed” (Zimmerman and Nardi 2006). In their study of cyberinfrastructure’s scale, Ribes and Finholt illustrate the tension between “today’s requirements [and] tomorrow’s users” (2007). Requirements elicitation is based on the current context, and is not forward looking, yet they rightly insist that it would be inappropriate to abandon requirements gathering altogether. The challenge is how to then make "transformative scientific progress" in the long-term while meeting short-term funding deliverables and without being able to see future users.

The problem is that co-evolution occurs between the social context of the cyberinfrastructure and the design of the cyberinfrastructure itself (Kling and Scacchi 1982). Systems are built for current users, who then respond and change under the influence of the system. The system may then be redesigned in response to the transformed users, and so on iteratively. This is exactly the phenomenon observed in infrastructure, that it "Is fixed in modular increments, not all at once or globally" (Star 1999). Yet this represents a tremendous challenge for those hoping to grow the increments of cyberinfrastructure, as they must decide upon the requirements that their incremental piece should fulfill.

In the literature, "scientific cyberinfrastructure" sometimes refers to the whole ecosystem of software, other technical artifacts, and social components that work in concert to support scientific work. Other times, the term refers more precisely to the middleware layer of the software stack that supports scientific software. For my purposes, I am interested in the methodology used to create the software aspects of scientific cyberinfrastructure, including middleware and the application level that interacts with users. For this reason I will generally talk about producing scientific software, rather than cyberinfrastructure. This dissertation is not focusing on the *use* of scientific software to support scientific work. Rather, I am interested in the sociotechnical system that supports the *creation* of scientific software. That is, I will consider the organizational practices, technical infrastructure, and social norms that go into *producing scientific software* at a distance. This parallels the definition of scientific cyberinfrastructure given earlier, which considers the same processes and objects but as applied to the smooth operation of scientific work. Certainly there are overlapping goals and actors, but I am focusing on producing tools for scientists rather than producing scientific knowledge itself.

I propose to focus on that process of handling requirements for building scientific software. The IEEE defines requirements engineering as the systematic handling of requirements (Abran and Moore 2004). For my purposes, *requirements engineering is the systematic and ad-hoc handling of requirements.* This expands beyond a strict view of formalized or systematic approaches to also include requirements that may be worked out and (at least partially) agreed upon, yet left undocumented. Requirements engineering involves eliciting, negotiating, managing, updating, correcting, and deleting requirements. Software engineering research has studied requirements engineering extensively, and will inform my approach. Furthermore, recent studies have begun elaborating on software engineering processes and software requirements as they play out particularly within scientific software.

1.3 Particularities of scientific software

While a good deal has been learned about software engineering practices in industry, the study of software engineering as practiced for computational science and engineering is relatively young. Research-oriented organizations that do this software development may have little or informal background in software engineering, and the specific needs of each scientific domain have implications for software design as well. The recent International Conference of Software Engineering workshops on Software Engineering for High Performance Computing (2005-2007) and Software Engineering for Computational Science and Engineering (2008-2009) have begun to investigate this area and lay out a research agenda.

Over the past five workshops, participants have identified a number of ways that doing software development work within the computational science context is unique. These are summarized below:

- Requirements are complex (e.g., a mathematical model)
- Requirements are dynamic (i.e., scientific discovery and dead-ends)
- Uses complex and powerful computer hardware
- Testing is important, and especially challenging
- Some software is produced as “Kleenex code” (i.e., one correct run is sufficient)
- Optimization may determine usability
- Software utilized by both casual and power users
- Scientists may do development work without formal software engineering training
- Scientists may code out of necessity, not desire and careful attention
- Scientific productivity is what matters most ultimately (Carver 2007; Carver 2009b)

These distinct properties of scientific software influence various aspects of the software engineering process, and more research is needed to tease apart the affects. Note that some of these are more technical factors, such as the importance of optimization or the complexity of the software requirements for mathematic modeling. Others signal the social side of the story, such as scientific productivity being the ultimate priority driving work on scientific software, and scientists doing this work with less formal training in software engineering. Next let's turn to some approaches for addressing these factors, and further research elaborating on software engineering.

Often computational scientists end up doing significant programming work to support their research. Smart people with a strong quantitative background can learn what they need to know in an ad hoc way, asking colleagues as questions arise or making use of online resources (Easterbrook and Johns 2009). Yet their programming knowledge may lack key ideas or refinements that a more systematic instruction would include. Wilson noted this issue among scientists at the University of Toronto, and created a hands on course about "Software Carpentry" to address the issue (Wilson 2006). The course was intended to be intensely practical, covering the low-hanging fruit of technologies and techniques that will yield significant productivity gains. In fact, Wilson reports that students were 20% more productive in their software work after taking the course. His experience indicated that the following topics were most beneficial:

- Version control software for synchronizing and archiving code
- Making builds repeatable using configuration management software
- Scripting for data munging
- Debugging processes and tools, especially integrated development environments
- Systematic and automated testing
- Continuous integration of code changes
- Processes for prioritization, estimation and tracking (Wilson 2006)

A second approach for allocating the software engineering work is to include computer scientists as collaborators on the project. Computer scientists will likely be familiar with many of the techniques and tools mentioned above, and will take advantage of those productivity gains. For much of the past century, demand for simulation and sophisticated computation has driven computer science research and development (Edwards 2010b). Computer science research, in turn, has enabled computation and data analysis at levels that support new types of scientific inquiry. This intertwining of goals is reflected in Carver's lists of distinctive aspects of software engineering for scientists, which notes that optimization is of critical importance for computational scientists

(Carver 2007; Carver 2009b). It is not uncommon for optimizing a program to take more time and effort than simply achieving the first correct implementation. This sort of optimization work can be both interesting and valuable to computer scientists, complementing their own research goals. Furthermore, the National Science Foundation and other funding agencies value this sort of cross-disciplinary collaboration that advances research in both fields and spreads funding dollars further by using computer scientists to help with the implementation work for computational scientists.

However, there are also a number of challenges with this sort of collaboration between domain scientists and computer scientists, as Weedman and Lawrence have both pointed out (Weedman 1998; Lawrence 2006). In addition to the normal challenges of working across groups that use different languages, this sort of inter-disciplinary work organized around a single artifact requires the negotiation of research priorities. For instance, sometimes domain scientists need a reliable tool that will allow them to do their science work easily and without disruption. However, computer scientists receive little reward for the debugging and user interface work that would take a prototype to the stage of being production-ready. For this reason, some partnerships between computer scientists and domain scientists may work better than others. Often the partnerships lead to prototype software systems that the computer scientists publish about, but are also difficult to use and unreliable for domain scientists. At other times, the partnership can be mutually beneficial. When developing "Kleenex code" software, where one correct run will suffice, computer scientists can be fruitfully engaged without worrying about spending a lot of time prettying the code and making it reliable or reusable under different conditions. Computer scientists can benefit from the algorithmic innovation needed to optimize the execution time to get one successful run, while domain scientists acquire the data that they need as output from the run. In general though, partnerships between domain scientists and computer scientists are an area of potential tension, even among kind and considerate collaborators, as different priorities pull individuals in different directions.

A third approach is to enlist a team of software engineers to develop software for the scientists. This development team could either consist of internal staff that regularly do this type of work for the institution, or could be hired externally. Segal's study of a group writing embedded software for an instrument is an example of this strategy (Segal 2005). She notes the tension around developers wanting to resolve requirements in order to proceed with their waterfall-style development methodology, but the research scientists kept shifting requirements as the project continued. This feature of researcher/developer coordination is a common challenge for such projects. Some of the scientists in her studies described "the difference between emergent and up-front requirements as illustrative of a culture gap between scientists and (the more traditional) software engineers." Segal goes on to recommend that an Agile methodology might better fit the scientific context.

Agile software development methods emphasize informal processes and iterative development that evolves requirements and solutions. The Agile Manifesto founding the style highlights four values: "1) Individuals and interactions over principles and tools; 2) Working software over comprehensive documentation; 3) Customer collaboration over contract negotiation; and 4) Responding to change over following a plan" (Cunningham 2001). This iterative, lightweight approach may better match the dynamic nature of research by expecting and allowing for requirements to shift substantially throughout the development process. Furthermore, Segal noted the team's challenge in constructing a shared understanding around requirements and expectations for the software, perhaps due to inadequate documentation. One assumption she discovered was "that project documents suffice to construct a shared understanding, and our evidence is that this simply isn't true" (Segal 2005). Agile addresses this challenge of shared understanding by leaning upon informal interactions, but this can be especially difficult for distributed development projects to achieve as we will see in more detail below.

Segal believes that Agile software development methods, and particularly paired programming, could help with both the iterative demands and documentation challenges of software development for scientists. Scientists and developers could pair together to work on code at times, developing a more informal shared understanding of the software. This would also address problems that the team experienced about losing expertise when some developers left the team. For dynamic projects that assemble different configurations of people it may be helpful to overcome losses with "promiscuous pairing" that mixes different pairs to further spread knowledge. Noting the logistic challenges paired programming can impose, Segal concludes that even "informal (unminuted), regular, frequent, face-to-face meetings" would be a low-burden way of "constructing and maintaining a shared vision" (Segal 2005).

The Met Office is the United Kingdom's national weather service, and is involved in meteorological and climate science research in addition to operational weather forecasting. Easterbrook and Johns studied software development activities at the Met Office's Hadley Centre, which focuses on climate change research. They attributed the success of this software development environment to the informal interactions and practices that they have established (Easterbrook and Johns 2009). Developer and scientists are collocated in the same building, and take advantage of the interactions this affords:

> Scientists at the Met Office use a number of different strategies to maintain a shared understanding of the software. Formal design documentation exists for the [suite of FORTRAN routines in the Unified Model], but is only sporadically updated. The scientists working on the model rely heavily on face-to-face communication, together with a large number of 'informalisms.' (Easterbrook and Johns 2009)

This interaction translates into a work style: "developers had a strong shared scientific background, adopt an informal, collegial management style, with a culture of member participation and shared responsibility" (Easterbrook and Johns 2009). The centre takes advantage of its stable workforce by cultivating configuration managers who "become the local experts for knowledge about how to configure the model for particular runs, and keep track of the experiments performed with the model." Rather than relying primarily on shared documents, they intentionally use individuals as repositories of expertise. Yet even in a relatively open site that includes many opportunities for informal exchange in addition to electronic and code documentation, there are chances for misunderstandings:

> we noticed some evidence of ontological drift (Robinson and Bannon 1991). For example a proposed glossary entry on "bit comparison" on an internal newsgroup revealed some people took it to mean the regression tests, while others took it to mean the platform configuration comparisons. (Easterbrook and Johns 2009)

Ultimately, Easterbrook and Johns believe the success of the scientist developers at the Met Office is due to them having a flexible methodology that they have customized to fit their needs. In its current form it has taken on some aspects of the work environment recommended by the Agile Manifesto, relying on informal interaction for communication. This has only been enabled since the Met Office moved to a new building where everyone was collocated, a move that employees overwhelmingly favor for their work (Easterbrook and Johns 2009). The key for software engineers at the Met is not just their proximity to other software engineers, but also to the scientists. This allowed the benefits of informal communication in an ongoing way between scientists and software engineers, allowing them to collectively develop, clarify, and refine the requirements the software should fulfill for the scientists. Many scientific development environments do not have such an intimate geographical configuration, instead scattering scholars and engineers across national laboratories, supercomputer centers, and universities (Finholt 2002; Olson et al. 2008a). This suggests it will be worth considering the global software development literature to see how distributed software engineering affects the scientific community.

The Global Software Development Handbook seeks to systematically distill recommendations and best practices from the literature (Sangwan et al. 2006). The authors discuss implications of distributed development on the planning process, such as including team leads from each site in the requirements elicitation process to facilitate knowledge transfer and more quickly get remote teams up to speed. They also recommend ensuring that appropriate people are part of each requirements stage, whether or not they happen to be collocated. Citing Herbsleb and Mockus, they urge software engineers to be particularly diligent in getting as many requirements as possible right at the beginning, since change requests "need 2.4 times longer to resolve in distributed environments as compared to collocated Projects" (Herbsleb and Mockus 2003).

Requirements engineering has emerged as one of the key challenges in global software engineering (Damian 2007), with key issues: How do stakeholders share knowledge and iterate on understandings of requirements? What are effective communication strategies for accomplishing this?

In suggesting important research challenges for global software engineering, Herbsleb highlights a few challenges for requirements engineering (2007). First he points out that research has focused on eliciting, analyzing, and managing requirements.

This needs to expand to requirement negotiation (ongoing throughout a project), which is much more difficult to do in a distributed context. Research on distributed work has shown that distributed teams have greater quantities of conflict (Hinds and Bailey 2003), and that negotiating common ground was a (rare) instance in which video helped non-collocated team members (Veinott et al. 1999). Research on distributed groups has suggested that technology and/or distance can create an "us vs. them" effect in which individuals make decisions based upon whether someone is collocated, rather than the most rational decision (Bos et al. 2004; Batcheller et al. 2007). Negotiating requirements may be significantly complicated by distance. Herbsleb encourages research to help understand how much negotiation might be needed: Can we predict requirement stability? What factors make it difficult to agree upon and maintain a common set of requirements? Which stakeholders need to be engaged in the process? Finally, Herbsleb urges expanded research on the effects of different communication media upon requirements communication. What are the communication needs in different aspects of requirements engineering, and how do the affordances of different communication media meet these needs? While this dissertation does not focus on the process of distributed software development, it will address the questions about the stability of requirements within scientific software, the engagement of different stakeholders in requirements work, and communication practices that facilitate requirements gathering and refinement.

1.4 Requirements engineering and scientific software

Requirements engineering is widely recognized as a critical part of building software even outside of global software engineering. In his book on "Rapid Development," software engineering guru Steve McConnell is enthused about the potential gains in development speed through better requirements management:

> Imagine what the combined effect would be if you could reduce the number of [requirements] changes from 25 percent to 10 percent and simultaneously reduce the cost of each change by a factor of 5 or 10. Rapid development would be within your grasp. (McConnell 1996)

McConnell is excited by data that show that 25% of requirements change in the course of a project. Getting requirements right the first time saves 50 to 200 percent in costs over requirements that are added or changed during construction and maintenance

(Boehm 1988). If requirements really must change as an inherent part of developing scientific software, it is worth seeing how we can minimize this and make requirements change less costly. Indeed this is a critical part of the study on software design and cyberinfrastructure urged by the NSF (Ad Hoc Committee for Cyberinfrastructure Research Development and Education in the Atmospheric Sciences 2004). Requirement engineering is both an element of problem definition that must occur as a first step of software design, and also recurs throughout science software projects as requirements change and must be integrated into the design and work plan.

Despite admonitions to get requirements right the first time, requirements remain difficult to specify correctly from the beginning. In practice, requirements are always updated, added and deleted throughout the software engineering process. Ronkko et al. suggest that we should stop thinking about missed requirements as a mistake, but rather as simply a part of the process as software engineers iterate through requirements as they also iterate through the implementation of the software (2002). However, it is clearly beneficial to minimize the frequency and cost of requirements changes. While some amount of change seems inherent to all projects, additional changes due to a rapidly changing domain like science could aggravate this challenge.

Consider the processes that are part of requirements engineering. The IEEE's Guide to the Software Engineering Body of Knowledge (SWEBOK) breaks down software requirements processes into the following activities, acknowledging that often projects will iterate between these:

- Requirements elicitation (also 'capture,' 'specification,' or 'construction' with slightly different philosophical orientations (Jirotka and Goguen 1994))
- Requirements analysis (including requirements negotiation)
- Requirements specification (documenting requirements explicitly)
- Requirements validation (reviews to ensure stakeholders agree on requirements) (Abran and Moore 2004)

My study here examines these processes, with an overarching research interest in the requirements engineering practices in projects developing software systems *for science*. However, the unit of the "project" may not include a sufficiently broad scope to capture the phenomena of interest. The vision of a project often begins to form before the project as a "thing" comes to exist. Key individuals begin discussing and defining a problem with some ideas about how to address it. Additional people are enrolled into the discussion. Eventually some funding is acquired, roles are formalized, and the formal project is launched. But by that time the vision has been, at least temporarily, fixed. So I intend to include in my study this early alignment of key people as the idea of a project is born, a process that is underemphasized in typical descriptions of requirements engineering (Wiegers 2003; Abran and Moore 2004). For linguistic convenience I will continue to use the term project, but it should be understood that I use this term loosely and include the fuzzy processes at a project's inception before it has formally launched.

1.5 Stakeholder engagement in requirements engineering

Software requirements elicitation – or the communication of needs from scientists to developers – is a particularly interesting and challenging problem here. As Wiegers writes, "Software development involves at least as much communication as computing, yet we often emphasize the computing and neglect the communication" (2003). Indeed, "capturing" requirements from stakeholders is difficult to the extent that they seem elusive, and the analysis of the captured requirements usually invokes negotiation among the stakeholders. Stakeholders are not merely the users/customers. In industry, stakeholders may include users, customers, market analysts, regulators, and software engineers (Abran and Moore 2004). In scientific software development, they could include scientists (faculty, research scientists, and graduate students), funding agencies, software engineers, and members of other projects (wishing for interoperability). So while I initially used "stakeholder" colloquially, a more precise definition put in the context of software requirements analysis helps illustrate the breadth of people who are entailed:

> A stakeholder is a person or organization who influences a system's requirements or who is impacted by that system. (Glinz and Wieringa 2007)

In software with the ambition of scientific cyberinfrastructure – intended to transform a discipline – this suggests a large number of individuals who would be impacted by the system. Not all stakeholders are likely to be identified as such. For instance, other software projects may need to produce files that are compatible with the system being designed, making users and developers of these other projects stakeholders in the design of the system. Some people may be identified as stakeholders, yet not included in the requirements engineering process – perhaps being deemed too peripheral to the project's core goals. Others are "negative" stakeholders whose interests are harmed in some way by the software project. The inclusion and exclusion of stakeholders is a fundamental process for directing the course of projects.

From the perspective of Actor-Network Theory (ANT) (Law 1991; Latour 1993), this is the process of arranging a network of (human and non-human) actors. Networks do not arise ex nihilo, but rather "new networks emerge out of already existing ones" such that "defining a beginning is a necessary but 'artificial' analytical operation" (Stalder 1997). In an emerging network, an intermediary is introduced that unites the new network of actors. In our case here, this would likely be an initial grant or the name of the new software project. Around this, actors align themselves and bring "more/different actors for the network's own interest" (Stalder 1997). As the network develops, actors continue to be brought into or out of the network, in turn shaping the network. These actors include the stakeholders we are interested in, and this process of assembling a project is well described by ANT's depiction of projects birthed from existing relationships, that co-evolve with the group of individuals engaged in the project.

Yet "actors" or even the more specific term "stakeholder" is an abstraction, glossing over the variation in stakeholders roles and in the types of stakes that they hold. I expect that this variation is important in directing the way that the project is formed, who exactly becomes involved, and how they impact the product to be created. For the purposes of this study, I need to open the black box around the role of stakeholder. This leads me to my first research question, which is intended to unpack and detail the stakeholders: ***Who are the stakeholders in scientific software projects, how do they become engaged in the project, and what role do they have?***

1.6 Product requirements and process requirements

The SWEBOK Guide helpfully maps out the types of requirements in which these stakeholders are interested (Abran and Moore 2004). They emphasize that a fundamental property of requirements is that they are *verifiable*, and quantifiably so if possible. This is to encourage specificity, preferring "the system will be available 99.5% of the time" over "the system will be reliable." Even so, the aforementioned requirement is an "emergent" property, which cannot be addressed by a single component of the system but only by all components acting in concert.

Wiegers breaks down requirements by first splitting off business requirements, which "represent high-level objectives of the organization or customer who requests the system" and are often captured in a "vision and scope document" (2003). Business requirements may also be referred to as the project's objectives or goals, and include the desired outcomes. For example, such an objective may be to become the default software framework used in a particular scientific community. These are separate from user requirements, which "describe user goals or tasks that the users must be able to perform with the product." User requirements may be represented with use-case scenarios. For instance: "the user shall be able to find the dataset they wish to download with a search feature and the system will send them an email with options for downloading the dataset when the files are ready."

Both business and user requirements must then be translated into specific requirements indicating how the software is to be built. These requirements may include both product parameters, constraints on the software that is to be developed, and process requirements, constraints on the software design process. For example a product requirement may be that the software needs to allow the storage of output files in a particular format like NetCDF, the Network Common Data Form, which is commonly used to store climate datasets and other array-oriented scientific data (Unidata 2010). A process requirement may specify that the code should be tested with unit tests on all supported platforms, and include the software team should have annual meetings with an advisory board. Within product requirements are both functional requirements, or capabilities, and nonfunctional capabilities, or quality requirements. For instance the example above about outputting data in a particular format is a functional requirement. Nonfunctional requirements would specify that the output must be completed within a certain amount of time, and that the code must allow alternate output formats to be added easily.

Just as we saw how scientific software can be framed as infrastructure that includes people, institutions, standards, etc., the SWEBOK Guide also acknowledges the broader system surrounding software. So in addition to software requirements, we can think about "system requirements" which includes the requirements of the entire system to outside stakeholders (Abran and Moore). Software requirements represent the needs stakeholders have for the software. System requirements represent the needs stakeholders have for the entire system, which I will define as a sociotechnical system that includes software and related people, institutions, standards, etc. For instance, a system requirement could be to produce research output as publications at a certain rate and to acquire funding at a certain level. The software itself is not to fulfill these requirements, but does so in combination with researchers, word processing software, administrative staff, a library, etc. This dichotomy between software and system requirements helps us think about the requirements for the software specifically versus the entire ecology of activity that comes along with it. Business requirements often must be satisfied with not just the software, but with related activities that position the software in the world – like persuading people to use the software and encouraging other groups to adapt their software to be compatible with it. In summary, we can define requirements by the following types:

1. Business requirements – project goals, objectives, and desired outcomes
2. User requirements – user goals and tasks to perform with the system
3. Process requirements – specify the manner in which system development should happen
4. Product requirements – specify the properties of the system to be developed
5. Functional requirements – what the system should do (e.g., accept certain input files)
6. Nonfunctional requirements – how the system should do it (e.g., finish in under an hour)

Requirements may be at the *software* level, specifying what the software needs to fulfill. Or they may be conceptualized as *system requirements* specifying what the larger sociotechnical system needs to fulfill.

Groups developing software for scientists are well aware of the interrelationships and importance of these different types of requirements. Consider this explanation of the "Project Vision" contained within the Earth System Modeling Framework's requirements document:

> These project objectives describe the desired impact of the ESMF on the Earth system modeling community. Although some of these objectives are non-quantitative and difficult to verify, they still must be satisfied to consider the ESMF project truly a success. It is essential to allow them to influence project execution, since it will be possible to satisfy detailed functional requirements but fail to achieve the underlying objectives of our effort.

This team is keenly aware of its need to think holistically in accomplishing their business requirements, not merely design good software. This begs the question: what is needed to satisfy project objectives other than completing the functional requirements? For instance, a business requirement in the sciences may be to increase interdisciplinary collaborations and data sharing. Part of the answer is in building software with the right features, and making it usable to the multiple communities intended to use it. However, there is additional work in outreach to other communities, perhaps engaging key individuals in the design process to help them tailor it and get them to buy into the idea of using the system. This may mean traveling to certain scientific conferences to present the new features and capabilities to potential users. Beyond functional requirements for the software are all the requirements that are technical in nature: the product requirements. This is in contrast to the requirements that are process oriented, in terms of how the team goes about building the software system. So perhaps the ESMF document is hinting that the way they go about their software development work, and the activities that complement their software product, are critical Yet these are just my speculations – how in practice do teams building scientific software work to satisfy business requirements? "Business requirements" are a strange term in this non-commercial context, so I will refer to them for the remainder of this dissertation as "project requirements." This leads me to my next research question, in which I compare the work of making the software system itself and the other work in which the software team engages: ***How are business requirements addressed by product requirements? How are business requirements addressed by process requirements?***

The distinctive qualities of scientific software, reported earlier, suggest some possible answers to these questions. Scholarly publication and the generation of knowledge is of central importance both as an end in itself and as an important step to career advancement and obtaining funds (Kendall et al. 2008). This objective of producing knowledge could lead a research team to choose a higher risk technical approach that includes innovative computer science approaches, because the team could get additional publications out of that work.

Olson et al. in their "Theory of Remote Scientific Collaboration" categorize important outcomes for successful cyberinfrastructure projects (Olson et al. 2008b). Many of the items listed could be formulated as business requirements within some scientific software projects, depending on the specific vision of the project. These categories direct our attention to the unique types of requirements that may be expected for scientific software. They identify six types of outcomes that count as "success":

1. Effects on the science itself
2. Science careers
3. Learning and science education
4. Inspiration to others
5. Funding and public perception
6. Tool use (Olson et al. 2008b)

While the most obvious requirements may fall in the category of impacts upon science, their typology helps direct our attention to other requirements like the ability for the work to be recognized professionally and lead to career advancement. In addressing the role of business requirements, I will look for how requirements in these broader sets of categories are handled. They may be addressed through alternate mechanisms, divorced from formal and recognized processes of requirements analysis. For instance, a rhetoric of optimism (or even hype?) about the revolutionary capacities of a new system may be important for acquiring funding and encouraging adoption. This could result in a requirement for creating enthusiasm at some level within the scientific community, independent of the actual functionality of the system. This is similar to the quasi-utopian rhetoric used by the "systems men" who sought to consolidate power under comprehensive, organization-wide information systems (Haigh 2001). The phenomena of utopian visions are common around new technologies, and certainly serve a purpose. In fact, they may have helped push investment in business technology along until it started realizing gains. After lamenting the "productivity paradox" in the 1990s (Landauer 1995), businesses finally began to realize productivity gains from technology in the 2000s (Brynjolfsson 2003). Perhaps it just takes a long time to harmonize the co-evolution of social and technical systems, and hype/optimism/vision may be an important requirement to drive work along in the interim.

1.7 Changing scientific knowledge and practice

Requirements negotiation is an important activity that may surface discussion about how the project's objectives should be interpreted and mapped into specific functional requirements. Such negotiation often happens in meetings specifically dedicated to working such issues out. Mapping project objectives onto other activities – like writing a paper or meeting with another team – is also important, but may occur in a more scattered way among individuals and across meetings. Teams developing scientific software are often working across multiple laboratories scattered geographically, complicating these requirements engineering processes. Damian and Zowghi map four areas known to be challenging for this sort of "global software development" (cultural diversity, inadequate communication, knowledge management, and time difference) onto specific challenges for requirements engineering. They found that the four areas all contribute to greater difficulty in achieving a common understanding of requirements (2003). Proactively "reducing ambiguity" is important for all activities within global software development teams, as such teams do not have access to the informal interactions that often handle ambiguity (Sangwan et al. 2006). When it comes to specifying requirements, this implies a particular importance on clearly specifying requirements. Yet as we saw from the research on software engineering for computational science, requirements seem to be especially fluid for science which is fundamentally involved in discovery and exploration.

Therefore, I am interested in investigating the ways that requirements change in the sciences, and whether this poses a particular problem for software development efforts. Requirements seem to be impossible to get right the first time, and initial requirements mistakes/omissions as being very costly for a project. Unstable requirements are one of the two most common causes of runaway projects, very expensive to fix during production, and missing requirements are the worst of the bunch. This seems like a central problem for software that seeks to transform the way science is done, making it very hard to predict what requirements will be needed. Furthermore, scientific needs may shift with new discoveries, uncovering a whole new set of requirements that were missing from the initial requirements analysis. This leads to me a research question to detail requirements changes as related to science changes: ***Has scientific knowledge or practice changed in ways that affect the software? How so? And has the scientific software impacted scientific knowledge or practice? How so?***

This is an important and complex pair of questions, but I believe that it is important to begin to address the issue of the mutual influence of science on scientific software and software on the science. This includes relatively straightforward cases like the demand for higher resolution climate modeling among scientists. This trend creates new demands for modeling software that can run quickly in massively parallel computing environments, servers that can share very large datasets, and visualization tools that can churn through vast amounts of data to generate images. Inversely, I want to know whether data sharing systems are allowing new types of scientific intercomparisons that were previously impossible, and what that means for the types of research that can be conducted. This should allow me to begin to address the dynamics and interconnections between scientific and technological advances and how they affect requirements engineering activities.

Chapter 2

Methodology

This chapter includes three sections. First, it provides a rationale for choosing the three case studies that I am investigating in this dissertation. Second, it provides a description of the perspective, methods and processes used in conducting the research for this thesis. Third, it provides an overview and history of each software case study, which is intended to orient the reader to the three projects and provide a context for understanding the particular situations and aspects of the projects addressed in subsequent chapters.

2.1 Rationale for choosing case studies in climate science

To address my research questions about requirements engineering in science, I chose to focus on three software projects within climate science. There are several reasons for this choice. First, climate change is a pressing societal problem (Pachauri and Reisinger 2007; Richardson et al. 2009), and while I hope that my work helps us better understanding scientific software in general, I would be particularly pleased if it were to benefit efforts to understand and respond to climate change. Second, climate scientists are major users of computer infrastructure and need to develop software to support their scientific work. As such, they are frequently engaged in design work and in managing requirements to figure out what software will maximize hardware resources and expedite science. They should provide a rich set of data about software development and requirements processes. Third, the atmospheric science community recognizes the need to support a "requirements analysis process [that] bridges the gap between technical collaborators and Geoscientists" (Ad Hoc Committee for Cyberinfrastructure Research Development and Education in the Atmospheric Sciences 2004). They also recognize that they need to support the whole software life cycle as the field has tended to prioritize "innovative development" and has "[neglected] funding for the full software life cycle, including hardening and deployment" (ibid.). This signals a community receptive to feedback about their requirements processes. Fourth, and not insignificantly, I have experience studying this domain as part of other research efforts and have existing contacts within the community that provide me with some domain knowledge and make it easier to gain access to informants and documents.

As the debate about global warming and what to do about it have heated up, the Intergovernmental Panel on Climate Change's (IPCC) assessments have been central to public discourse. Assembling such a vast report with care has commanded much attention from climate scientists, even setting the rhythm of climate science research. The IPCC produces Assessment Reports periodically to compare and summarize the current state of knowledge about climate change, with the most recent report published in 2007 (Pachauri and Reisinger) and the next report expected in 2014. This demand for a systematic intercomparison of scientific results led the climate science community to develop software with timelines synchronized to the deadlines of the IPCC. Here we pick three software projects that support different aspects of the chain of activities that happens during climate model intercomparison activities. The three projects selected range from 10-19 years old, and have from 6-25 full-time employees working on them. (However, we will see how such things as project age and number of team members are surprisingly difficult to measure.) The Earth System Modeling Framework (ESMF) aims to support modelers in the process of developing climate modeling software and facilitate the coupling of different model components. The Earth System Grid (ESG) provides data portals that simplify the process of moving large datasets from servers to scientists who wish to use the data. The National Center for Atmospheric Research (NCAR) Command Language (NCL) is a post-processing utility produced by NCAR that does file input/output, data analysis and visualization. It serves as a graphics tool that uses custom scripts to produce images from datasets – such as those retrieved from ESG or generated by a model using ESMF. These three projects are both important in their own right for the climate science community, but they are also representative of classes of projects that are important for scientific work more generally: a modeling framework, a data portal, and a post-processing tool. These three cases also represent teams of software engineers doing software development full-time for the research community, which stands in contrast to other studies of scientific software that have emphasized scientists doing software development part-time as non-experts (Carver 2009b; Ribes and Lee 2010).

2.2 Research design

For the range of process-oriented questions I propose to address, detailed, descriptive data are needed to answer my questions about "how" requirements are handled when developing scientific software. Qualitative methods are well suited for eliciting the rich data needed to understand the complex dynamics involved in software development. My two case studies with external sponsors, ESG and ESMF, produce many proposals and reports, and all three case studies have large websites. Each project has a collection of email lists that they use, and document meeting minutes. This set of written records is a great opportunity for document analysis, with glimpses of how the projects represent themselves to others and how they communicate internally. Observation would be a great method to see how the work is actually done, irrespective of the filters applied when writing or recalling past events. However, the distributed nature of the teams and my personal constraints made it impossible to do large amounts of physical observation. I spent a few days with the ESMF team during a series of their meetings, but do not have a parallel experience with ESG or NCL. Much of the projects' lives are lived virtually through online environments, and especially through email, and I have been able to substantially observe each group virtually. However, my main method for data collection was with semi-structured interviews, where I guided participants through a set of questions related to their requirements work.

2.2.1 Document analysis

My document analysis consisted of reading archives of email messages, publications and internal documents, and website material about the projects. I particularly sought records of requirements documentation and proposals to funding agencies. Funding proposals and reports to funding agencies are an important location for some requirements to be represented system requirements that go beyond what the software alone needs to accomplish. So I have sought out such reports, strategic plans and vision documents. Most documents I simply used to enhance my overall understanding, especially of historical views that do not come out in interviews or observation. However, I did code an annual report for both ESG and ESMF, as well as one of ESMF's strategic plans. I did this in order to better understand the differences between how stakeholders and project requirements are discussed formally in contrast to the way my interview participants talked about them.

ESMF is especially open about its documents, so it was relatively easy to obtain these. Minutes from meetings of boards and committees are posted online, as are annual reports, strategic plans, and (some) funding proposals. I also was able to join their email lists: "esmf_core" for traffic among core team developers, "esmf_jst" for email with the Joint Specification Team (a larger body of people with interests in the ESMF), "esmf_support" for support requests and replies, and "nesii" for informal or administrative core team conversation. I also was able to obtain the original requirements document written in the early days of ESMF, and their ticketing system used to track requirements, tasks and bugs is open to the public.

ESG has some reports and proposals available on its web site, and I was able to view use cases and meeting minutes on a wiki-based tool that the team uses. However, my record of ESG documents does not track as far back in the project's history. I also have not had access to the internal ESG email lists, but have subscribed to the low traffic list that ESG uses to communicate to users.

For NCL, I am subscribed to the high-traffic "ncl-talk" email list that is used to support NCL users. Users ask questions, report bugs, and ask for features, and the developers (or other users) reply to their inquiries. I do not have access to their ticketing system for tracking tasks, and they do not produce reports and proposals like the projects with external sponsorship do.

2.2.2 Observation

Observations offer a direct look at the dynamics within software teams, removing the (conscious and unconscious) filtering that participants insert during interviews. Being distributed projects far from Michigan, I was not able to directly observe much. However, I did sit in on teleconference calls and have been to several project meetings. I have also read meeting minutes, a proxy for observation, and followed email lists that are a form of direct interaction among participants and hence an opportunity for observation. Although I listed email lists as a form of documentary data, they also function as a way to observe the daily work practices of the group.

I have sat in on two sets of ESMF meetings. First, I observed a community meeting and the advisory board meeting held concurrent with the American Geophysical Union (AGU) 2008 meeting in December 2008. More recently, I observed two days of core team meetings held just after the CCSM Annual Conference in July of 2010. I have also visited the ESMF team at their old offices at the NCAR Mesa Laboratory in June of 2009, and at their new offices at NOAA's Earth System Research Laboratory in May of 2010. At those visits I was able to hang out with the team, attend their weekly teleconference call, and interview several of the developers. I have also observed two of ESMF's weekly teleconference calls at other times.

For ESG, I have observed five teleconference calls, particularly ones in which they have met with the ESC team or with people from the Metafor project. Two of these have been demo calls, in which the developers are showcasing their work to other developers and to potential users.

I have not observed work for NCL other than following their "ncl-talk" email list that shows some of the (substantial!) time that developers are spending every day on user support.

2.2.3 Semi-structured interviews

Semi-structured interviews provided the largest portion of the data used in this thesis. During these interviews I used an interview guide (see Appendix A), but allowed conversation to drift (within bounds) as topics of interest were introduced. I sought to keep interviews grounded in accounts of specific events, rather than asking for conceptual or synthetic responses. In preparation for each interview, I reviewed documents and previous interviews that could give me specific questions to ask my informants or events that I could call to their attention as examples of what I would like to hear about. Other times I asked for a detailed account of a specific stakeholder or requirement over the course of the project. As I used the interview guide more, I adjusted the way I used it to focus on the questions that seemed to elicit the best responses.

In developing my interview guide, I reviewed the interview questions generously provided by Jorge Aranda (Aranda et al. 2007) and Jeffrey Carver (Kendall et al. 2008) in their studies of software engineering practices in science and engineering domains, adapting some of them to my purposes here. Aranda's study focused on relatively small groups of scientists working on software, who had few formal requirements processes. Aranda had a relaxed and relatively lingo-free style of questioning that I sought to emulate. For instance, to ask about prioritizing requirements he would say: "Who determines what you need to develop. Why?" He also inspired me to ask about risks, which are a way of getting at success/failure metrics. Carver's study was of high performance computing groups with an explicit goal of providing back recommendations about how to improve the delivery time of their software products. The Carver instrument asked questions in more formal terms, such as "To what degree did your project employ requirements management?" I found this formality less helpful, since it may not be the terms that the software teams use in their own way of thinking about it. However, I did like the way that the instrument systematically directed participants' attention to different parts of requirements engineering work. I sought to achieve similar coverage in my interview protocol, but while using less formal, lingo-laden terms. My interview protocol was also informed by the protocol that I had helped to develop as part of the Monitoring, Modeling, and Memory (MMM) project that studied data practices in scientific cyberinfrastructure projects. Although that protocol had focused on data-related issues, I borrowed liberally the more general questions about participants' background and work practices.

I transcribed, coded, and analyzed 41 interviews conducted with 28 different participants. Additional interviews that I did for the MMM project also inform my thinking in less direct ways. The participants are listed in Table 1 by their pseudonyms. These participants included stakeholders that are part of each project's requirements engineering activities – for instance, members of a Change Review Board. Note that some individuals are involved in multiple projects, and scientists were often users of several of the software systems. I looked for opportunities to interview individuals who are involved in multiple projects, and thus can offer multiple perspectives. These are indicated in the table. Those who could be using multiple systems that I was studying, but had chosen not to, also presented an important opportunity for me and I asked them why they refused to use the other systems of interest to me or why they used it in some token way rather than in the way it was intended. I intentionally interviewed some people multiple times, so as to develop better rapport with them, to hear how the project they are working on has changed between interviews, and to ask more specific follow-up questions as I understand the project more deeply.

Pseudonym	Project	Role
Andre	ESMF	Sponsor
Grace (4 interviews)	ESMF, ESG	Software team
Patrick (4 interviews)	ESMF, ESG, NCL	Software team
Juan (2 interviews)	ESMF	Software team
Edward (2 interviews)	ESMF	Software team
Martin	ESMF	User
Jennifer	ESMF, ESG	Software team
Rahul	ESMF	User
Leroy	ESMF	Sponsor
George	ESMF	Software team
John (2 interviews)	ESG, NCL	Software team
Elizabeth	ESG	Software team
Lars (3 interviews)	ESG	Software team
Denard	ESG	Software team
Tom	ESG	Software team
Marie	ESG, NCL	User
Rachel	ESG	Software team
Wen (2 interviews)	ESG, ESMF	Software team
Jordan	ESG	Software team
Albert	ESG	Software team
Frank	ESG	User
Ashley	ESG	User
Avi	NCL	Software team
Marina (2 interviews)	NCL	Software team
Melissa	NCL	Software team
Josh	NCL	User
Cameron	NCL, ESG, ESMF	User
Vena	NCL	User
Gary	NCL	Software team

Table 1: A list of the people interviewed across the three different projects.

2.2.4 Analytic approach

Before coding the interviews, I constructed a codebook of themes, concepts, and activities that I wanted to use to classify the data. I started with the codebook developed for the MMM project and extended it with codes (i.e., labels for sections of transcripts and documents) related to requirements engineering. For example, I coded for discussion of “stakeholders” or mention of “requirements engineering” or other types of requirements. This was not a static codebook, as I actively looked for emerging themes that seemed to be important even though I had not thought of them initially. For instance, I found “feedback” and “support” to be recurring topics, so I added them to my codebook. All interviews were transcribed and added to NVivo, along with some key documents or emails. NVivo is a software package that allows one to code documents, and then assists with analysis by allowing the ready assemblage of data based upon specific codes, or overlapping codes. Using NVivo both helped to organize the data in a central place and allowed me to easily assemble all the data about specific codes for analysis.

2.2.5 Validity

Checking my findings will help to validate my research. As I proceeded through my interviews, I posited some of my preliminary findings to interviewees to get their feedback about whether I understood a specific phenomena or relationship correctly. I would also just (discretely and anonymously) bring up an interesting comment that I had heard from another participant to get this participants reaction. I particularly sought feedback from interviewees who have experience in other projects, and can comment on whether my findings resonate with their experiences in those projects. I had anticipated running a focus group as my analysis wrapped up to get a fuller set of feedback from participants, and to stimulate discussion among them about my findings. However, I ran out of time to plan and carry this out, so I did one follow-up interview as a "member check" with one of my key informants. He verified most of my findings as seeming on track, although he did help complicate my understanding of power users to differentiate those that have a relationship with developers versus those who do not.

2.2.6 Limitations

The approach I have described supported a detailed description of how requirements engineering work in three scientific software projects. However, there are several limitations. First, I do not have a direct comparison case from industry. So my interest in contrasting industry requirements engineering practices to scientific ones could only be addressed satisfied indirectly through comparisons to the literature.

Second, I have sought to involve all types of stakeholders in my interviews. However, there is a risk that I have overlooked some or have certain stakeholders underrepresented. The snowball methodology exacerbates this risk, as it could perpetuate a sample biased toward certain types of participants. To address this problem I was proactive in identifying different types of stakeholders to ensure that they are represented in my sample. For instance, just a month or two before finishing this dissertation, I noticed that I did not have good firsthand accounts from ESG users, so I sought out two additional participants who gave great interviews and significantly shifted my understanding of how ESG is being used. There still remains a risk that I missed participants with important alternate perspectives, particularly those who are critical of the project or those who are “refusers” and know of the software system but intentionally do not use it. It is less likely that I would have encountered such individuals as I snowball among people who use the software or are mentioned in project documents and emails. This risk is partially mitigated by the overlapping pools of stakeholders across the three case studies, so while I was interviewing an individual primarily about one system, I would also ask about the other two systems. Using this approach, several times I talked to individuals who were enthused about one project, but would give me a more critical evaluation of one or both of the other projects.

One outstanding limitation in my sample that concerns me is that I have not had good access to interview and observe the ESG software team. ESG is a different type of project from ESMF and NCL in that it is a service-oriented web service, rather than one involving code that users download. Users are also much more intermittent in their engagement with the service, and do not have much commitment to ESG as a service over any other means of acquiring their data. These factors imply to me that ESG’s relationship with users and requirements processes may look different from the other projects, but I do not have a clear picture of how it does work within ESG.

Third, I have sought to explain how certain requirements processes happen but only have accounts of those events. That is, I did not have the time or access for detailed participant observation of the participants. Participants’ accounts may be filtered and provide a specific lens into their memory of how requirements engineering works on each project. I countered this by both asking for grounded, specific descriptions of events, and also accepting informants’ stories as just one perspective on the situation. I also used probes and follow-up questions to try to move beyond “public relations” answers to get more nuanced and detailed accounts of events. Ultimately, more observational data would be a more effective counter to participants’ biases. Prolonged virtual (i.e., online) and physical observation of the stakeholders in each case study would add valuable data, laden with my own biases rather than informants'. This would likely provide verification and complication of the understanding relayed during interviews.

Lastly, I also believe the research could benefit from further member checking now via focus group discussions of the findings presented in this dissertation. I have checked my understandings with my informants throughout the project, but now have much more concrete analysis to see discussion with informants. I was only to do one interview that sought to systematically check my interpretations.

2.2.7 Institutional Review Board

Because this research involves human subjects, I sought and received approval IRB from the University of Michigan Institutional Review Board (Health Science and Behavior Sciences, IRB # HUM00037406, approved 5/6/2010). I also received approval to reuse the Monitoring, Modeling, and Memory project's interview transcripts and to keep my transcripts in an anonymized form for others on my research team to use. All participant names that appear in this dissertation are pseudonyms, and some biographical details are changed or omitted to protect their identities. I have also revised the transcript as quoted here when particular speaking patterns, especially related to English being a second language, might make the participants identity easier to determine.

2.3 Overview of Case Studies

In the remainder of each chapter, I introduce the three software projects that are my case studies. For each, I will briefly overview the software, give some background on the project, and describe the team's current structure. More detailed descriptions will be included in subsequent chapters as necessary for the particular concepts being addressed.

2.3.1 Earth System Modeling Framework (ESMF)

ESMF is intended to assist model developers by making it easier to reuse code. The software is primarily targeted toward groups building climate and weather models that simulate the earth system by performing numerical calculations on a gridded representation of the earth. Earth system models work by coupling together software components that model different parts of the earth system, such as the atmosphere, the ocean, or land ice. Coupling the atmosphere and ocean thus involves using data output from the atmosphere component as input to the ocean component. For instance, the ocean needs to know the temperature and wind velocity in the atmosphere immediately above it. ESMF aims to make this code development easier in two ways. First, ESMF provides what they call an “infrastructure,” which is basically a library of utilities that model developers can use, such as functions that assist with data input/output to different file systems and a calendaring system that will help the model keep track of how it is stepping through time. ESMF developers implement the infrastructure, and model developers can reuse it rather than having to implement those features themselves. The second way that ESMF promotes reuse is with a “superstructure” that facilitates coupling of different model components. Model components that are implemented in ESMF can more easily be coupled to other ESMF components. The idea is that this makes it easier to reuse a successful oceans component in other earth system models, so that each modeling group does not need to develop its own ocean component. Or at least modelers could more easily mix components to create novel configurations of earth system models. ESMF is considered a framework because model components need to be (re)written to conform to this coupling standard. Figure 1 shows the Goddard Earth Observing System Model, Version 5 (GEOS-5) model as implemented within ESMF, illustrating how the framework couples different components of a GCM.

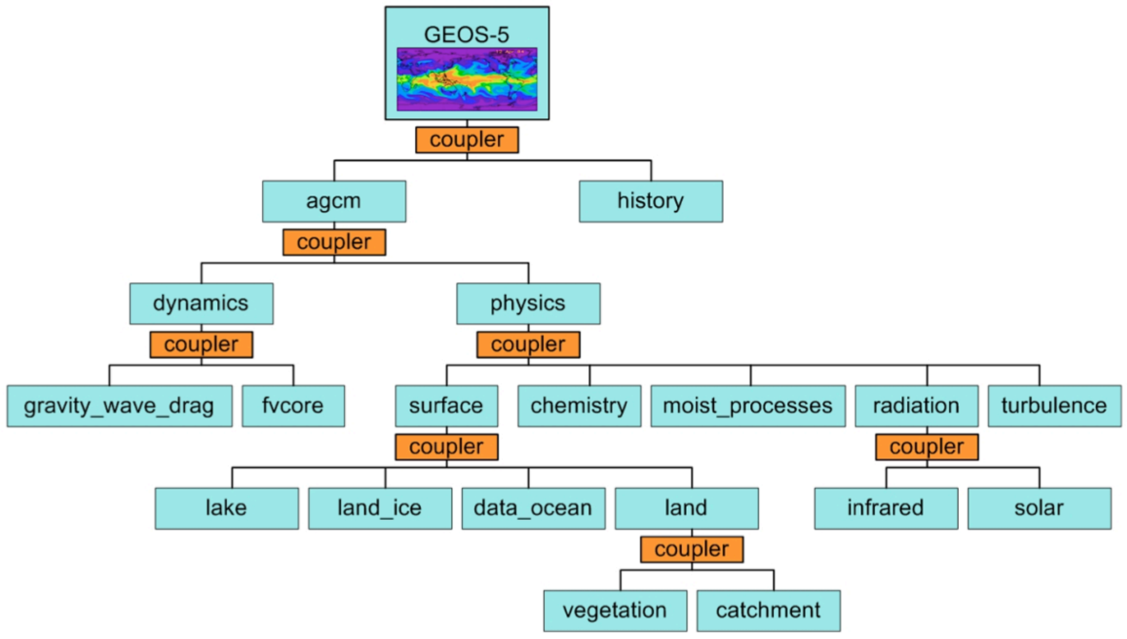


Figure 1: ESMF couples GEOS-5 model components.

ESMF's origins date to the late 1990s, when the National Science Foundation (NSF) appointed an advisory group to review software development practices at NCAR and concluded that NCAR's processes were not robust and efficient. These findings, along with other calls for better software infrastructure, produced an environment primed for proposals to improve software systems. At the same time, a working group of climate modelers started looking for ways to share code across models. This informal group produced preliminary requirements and a design for a modeling framework, and successfully competed to build the "Earth System Modeling Framework" in response to a National Aeronautics and Space Administration (NASA) call for proposals in 2000. During this initial NASA funding, model developers implemented GEOS-5 within the ESMF framework, which helped validate that ESMF would work for a major climate model. GEOS-5 was the first major modeling code to adopt ESMF, and remains a major success story for the ESMF team. Because ESMF started at the same time as the GEOS team did a major rewrite of their model, the GEOS team was able to utilize ESMF in a deep and systematic way. After the first three years of NASA funding, the ESMF team transitioned to a funding model that drew from multiple sponsors when it won an award from the Department of Defense (DOD)'s Battlespace Environments Institute (BEI). The National Science Foundation (NSF) helped by funding some ESMF developers who were NCAR employees. In years following, ESMF received major funding from the National Oceanic and Atmospheric Administration and the DOD to develop ESMF to be the framework for a joint NOAA/DOD initiative for a National Unified Operational Prediction Capability (NUOPC). When ESMF transitioned to multi-institutional sponsorship, both the sponsors and the project leaders were concerned that ESMF might be pulled toward prioritizing the needs of a subset of ESMF's growing community. So ESMF formed a Change Review Board (CRB) to manage and set the software team's work priorities. Managing the work through the CRB and a formalized governance structure is intended to increase transparency and help to balance decisions about prioritizing software requirements.

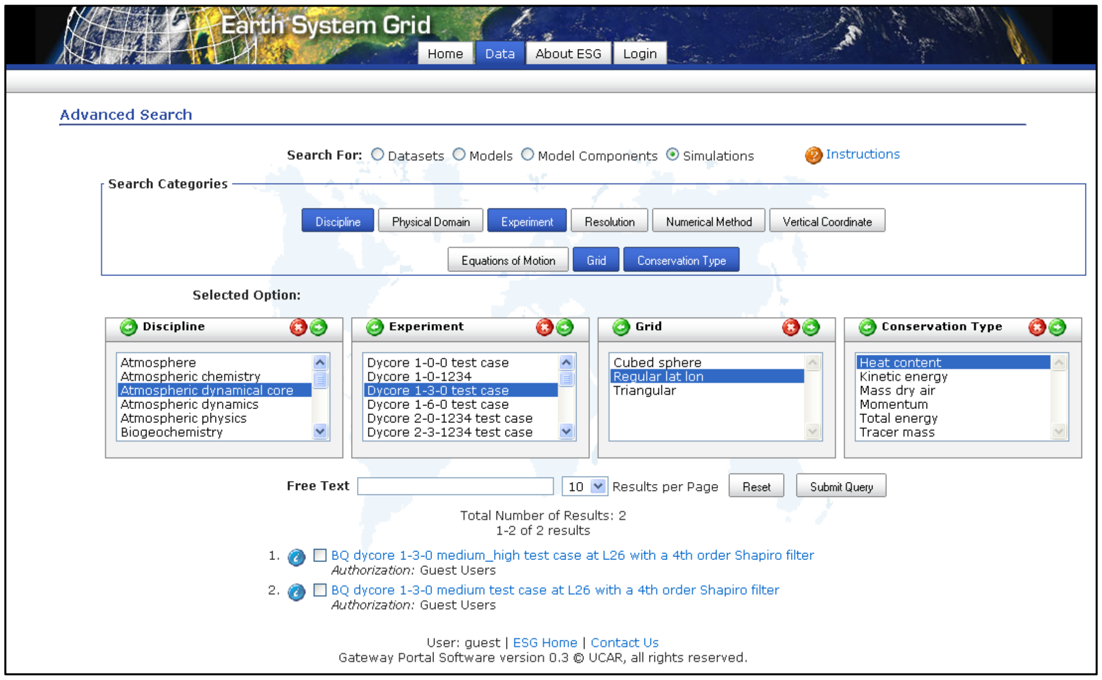
ESMF is open source and uses a single SourceForge code repository to manage its work. Anyone can access the code, although code contributions are regulated. Most users download ESMF to either help them build a model or because it is required to run a model that uses ESMF. Different models use ESMF to different levels, and the ESMF team has posted guidelines for evaluating the level of adoption. At the beginning of the project, the team wrote an extensive requirements document along with interested potential users. For the first few years of the project, this document directed development efforts. Eventually the remaining uncompleted items were absorbed into a ticketing system for tracking bug reports and feature requests, a system that is used to this day. The team holds weekly teleconference calls and makes extensive use of mailing lists for internal communication (the esmf_core list) and for communication with users (the esmf_support list). As of 2011, ESMF is preparing to release the first stable version of the framework that will guarantee a stable interface with backward compatibility for user code.

As of 2009 (the latest statistics listed), ESMF reports that 33 model applications have been rewritten using the ESMF framework. Each model could have many scientists running it as end users.

2.3.2 Earth System Grid (ESG)

ESG is a web portal used to access climate science data. Users can log in to a website, search or browse different data collections, and select the data they wish to download. ESG is unlike NCL and ESMF in that it is a service that users access. There is no ESG software that end users need to download or configure, and users do not write code in order to use ESG. In addition to hosting data, ESG hosts software such as modeling code and visualization tools for users to download. ESG makes is easier to access data because it takes care of finding the data and getting all the necessary permissions to access it. Architecturally, ESG is a federated system with multiple gateway nodes that each offer a different version of the web portal to users. Gateway nodes do not hold the data, but are connected to some number of data node servers. The nodes are federated, so that searching for data at a gateway node will reference the data stored in all the data nodes associated with that gateway. ESG works closely with the Earth System Curator (ESC) project to improve ESG's capacity to handle metadata about climate models. ESG already had metadata to describe the data via the self-describing

NetCDF data format (i.e., describing the name and meaning of different variables). ESC added metadata to describe the model that produced the data (i.e., specifying the model name and configuration). To demonstrate this capability, the ESC team set up an instance of the ESG gateway to host the data generated for and during a workshop comparing "Dynamical Core" models, which govern the physical interactions in atmospheric models. The ESG gateway was was intended to facilitate access to the data by all participants (see screenshot in Figure 2 below). Each dataset included metadata about model configuration, such as the grid size, grid type, numerical methods, or diffusion types. While mostly unused since the workshop, this portal has served since as a technology proof case for both ESG and ESC.



Search For: ( ) Datasets ( ) Models ( ) Model Components [✓ Simulations]

Search Categories

Selected Option:

Discipline	Experiment	Grid	Conservation Type
Atmosphere	Dycore 1-0-0 test case	Cubed sphere	[✓ Heat content]
Atmospheric chemistry	Dycore 1-0-1234	[✓ Regular lat lon]	Kinetic energy
[✓ Atmospheric dynamical core]	[✓ Dycore 1-3-0 test case]	Triangular	Mass dry air
Atmospheric dynamics	Dycore 1-6-0 test case		Momentum
Atmospheric physics	Dycore 2-0-1234 test case		Total energy
Biogeochemistry	Dycore 2-3-1234 test case		Tracer mass

Free Text [ ] 10 Results per Page

Total Number of Results: 2
1-2 of 2 results

1. BQ dycore 1-3-0 medium_high test case at L26 with a 4th order Shapiro filter
Authorization: Guest Users
2. BQ dycore 1-3-0 medium test case at L26 with a 4th order Shapiro filter
Authorization: Guest Users

User: guest | ESG Home | Contact Us
Gateway Portal Software version 0.3 © UCAR, all rights reserved.

Figure 2: Version of ESG gateway for the Dynamical Cores workshop.

The Earth System Grid began in 1999 as part of the Department of Energy's Next Generation Internet program. Three institutions led the effort, with principal investigators at Lawrence Livermore National Laboratories (LLNL), NCAR, and Argonne National Laboratories and additional collaborators at other DOE laboratories. These principals had observed that they were dealing with larger and larger datasets at their laboratories, and they a way to move this data around on national networks. So the initial vision of the ESG was very technology-driven, creating a prototype system to move large amounts of climate data around the network. They succeeded in making progress on this front, winning an award at the Supercomputing Conference in 2001. The Next Generation Internet program soon dissolved, so they turned to a new Department of Energy (DOE) effort, the Scientific Discovery through Advanced Computing program, to continue their funding. They successfully competed for an award to build ESG2, with an increased focus on scientific applications. The main goal was to distribute data from the Community Climate System Model and Parallel Ocean Program to those interested in using it. At this time, features were added to extract and handle metadata better, to include a catalog for finding datasets, and to increase security. The system was becoming ready to handle data distribution in a more complete, production-quality sense.

Concurrent with the development of ESG2, LLNL's Program for Climate Model Diagnosis and Interpretation (PCMDI) was responsible for distributing data as part of the Coupled Model Intercomparison Project (CMIP), which compares and evaluates modeling results and has an important role supporting the IPCC's assessment reports. When the IPCC asked PCMDI how they were going to distribute the CMIP data used in the upcoming 4th Assessment Report [60], PCMDI decided to use the ESG2. By 2004, ESG2 was in production mode, hosting 35 terabytes of data from NCAR's Community Climate System Model (CCSM) and other CMIP models. The data demand coincident with the IPCC process helped ESG become established as a major data distribution system, serving more than 130 terabytes to about 4,000 different users [65]. In 2006, the ESG team received a new award from the DOE, giving them five more years to continue their work, now as the ESG-Center for Enabling Technologies (ESG-CET). After this expires, they will likely pursue a new DOE award. The first few years of the ESG-CET award were used for more technology development, but the team is now focused on preparing for the next major model intercomparison, CMIP5, which will produce results for use in IPCC Assessment Report 5 (AR5). The ESG team now anticipates storing approximately 10 petabytes of data, a factor of nearly 1000 increase over CMIP3 in 2004. They also anticipate expanding with ESG gateways at major climate research centers internationally. These gateways will each have a catalog of the full set of CMIP5 data, and will allow users to search and choose files to download. Some 10-20% of the most frequently used data will be cached at each gateway node to speed access to these data, but the rest of the data will be kept in a data node at the center that generated the data. In preparing for CMIP5, the ESC team has built a system to handle metadata for CMIP5. To do so, they have coordinated with a similar group in Europe: the Metafor project. The Metafor team is developing a questionnaire to ask scientists about the characteristics of their models, and will ask each modeling group that participates in CMIP5 to complete the questionnaire. Metafor then transfers this metadata over to ESG, where it is ingested into the ESG databases and associated with the model data. There are multiple sites hosting data for CMIP5, but ESG is the sole system that provides broad access to the full set of data. This gives ESG powerful leverage in acquiring users and in making requests for modeling groups to conform to metadata standards. ESG expects a wide range of users of their CMIP5 dataset, ranging from climate scientists, to policy makers, to students and the general public. At the same time, ESG will continue to be used to distribute software and other datasets important to scientists.

ESG's principal investigators form a small executive board, which coordinates work across their different institutions. Different institutions have responsibility for different components of the software, permitting a low level of coordination across institutions that correspond to the loose coupling between components. Within a single institution, like NCAR, the ESG developers hold weekly meetings to coordinate work. Developers that need to coordinate with sister projects like ESC or Metafor will join those mailing lists and teleconference calls as part of their regular work too. ESC contributes to the development of aspects of ESG related to importing and handling metadata, but ESC also supports the integration of these same metadata schemes into other systems like ESMF, so that in the future models could automatically generate metadata rather than having to complete an onerous questionnaire by hand.

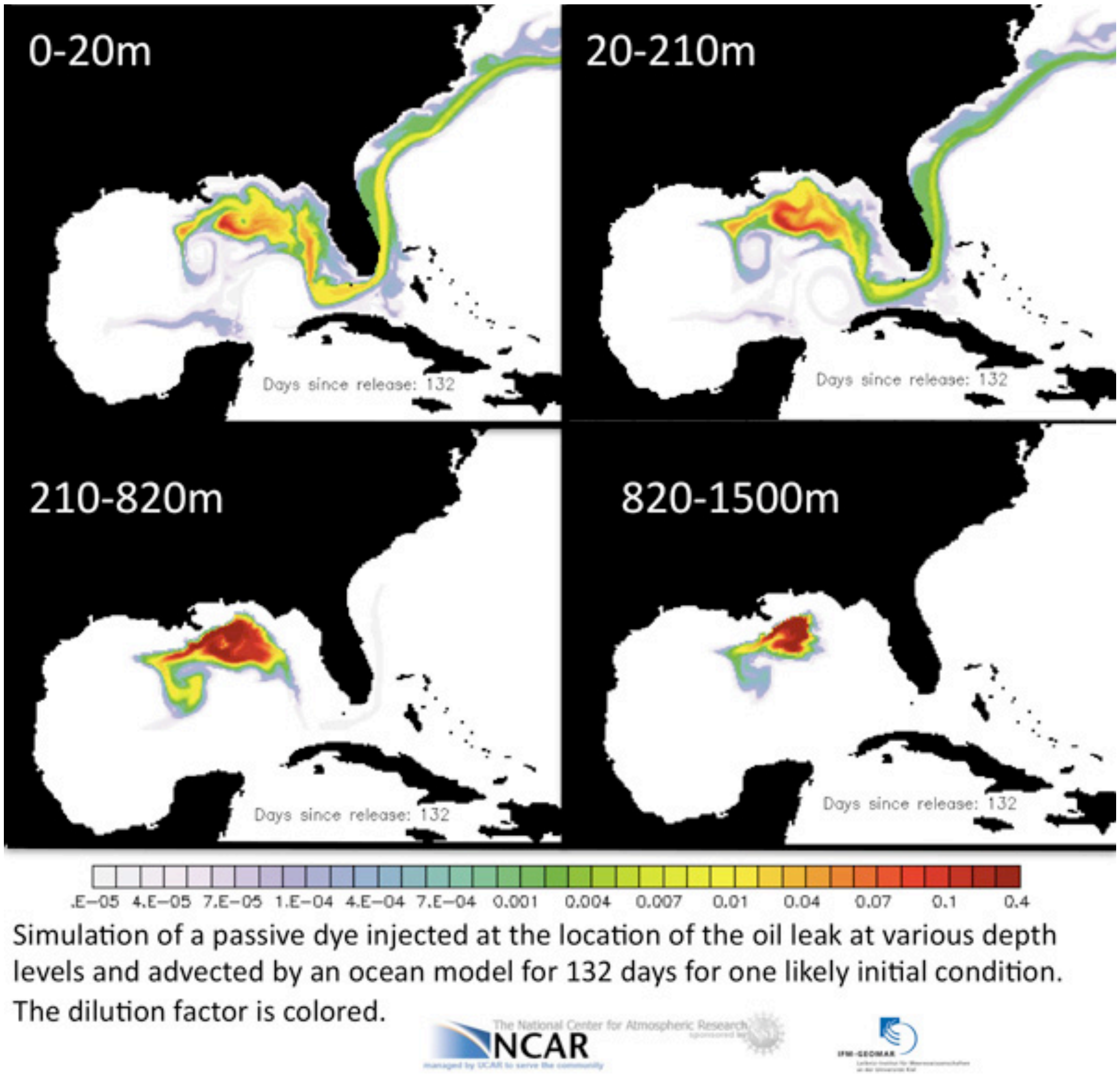
As of 2011, ESG reports 11500 users just at the NCAR portal, 1200 of whom have downloaded CCSM model data. The majority, 9700 users, have downloaded the post-processing tools NCL, PyNGL, or PyNIO.

A special note about ESG: My data about the ESG team is concentrated on the NCAR portion of the project which is in charge of building the gateway. I also have some data about the project overall, but do not have much about the ESG components being written at the DOE laboratories, which are the main sites of ESG work outside of NCAR.

Also, the NCAR portion of the ESG team has had some turnover recently. The main technical person left the team, and another key developer has changed positions as well. The project is under new technical leadership. I had trouble getting people to agree to interviews, and I suspect that may be due to the recent transitions and sensitivity about how the team is doing. The team is also under significant pressure to release a system to handle CMIP5 data soon.

2.3.3 NCAR Command Language (NCL)

NCL is a scripting language specialized for climate science and weather applications. Users write a script, and then run the script using the NCL interpreter. Depending on the contents of the users script, this could convert files between formats, perform some analyses and display the results, or generate graphics and plots for a publication. NCL is free and open source, and is supported by a small team of developers at NCAR. Being known as some of NCAR's visualization experts, NCL developers were recruited to help rush out some results of model runs simulating the dispersion of oil from the 2010 BP Deepwater Horizon well blowout. Figure 3, below, is included as a sample of the type of output that NCL generates. NCL is particularly known for its publication-quality figures, a feature of great value to scientists. NCL is both open source and free. It is difficult to compile, however, so the NCL team distributes compiled binaries for common computer platforms, and installs it directly on the supercomputers that many of its users utilize.


0-20m
20-210m
210-820m
820-1500m
Days since release: 132
.E-05 4.E-05 7.E-05 1.E-04 4.E-04 7.E-04 0.001 0.004 0.007 0.01 0.04 0.07 0.1 0.4
NCAR
The National Center for Atmospheric Research
IFM-GEOMAR

Simulation of a passive dye injected at the location of the oil leak at various depth levels and advected by an ocean model for 132 days for one likely initial condition. The dilution factor is colored.

Figure 3: Images produced by NCL of the 2010 BP Deepwater Horizon oil spill.

Visualizations have long been important for atmospheric science work, and the history of NCL dates to the early days of supercomputing (Middleton-Link et al. 1995). The 1960s saw the creation of some plotting routines that were shared among scientists at NCAR, and consolidated into a graphics package in the 1970s. At this time, NCAR began to offer its graphics package to the broader scientific community. In the 1980s, the package was ported to FORTRAN 77 and became known as NCAR Graphics. NCAR Graphics was just a library, so using it to generate images required writing a FORTRAN or C program. By the early 1990s, NCAR decided that this was an unnecessarily complex process and decided to write an interpretive language, or scripting language, that would allow scientists to more easily take advantage of NCAR graphics' capabilities. This nascent effort became known as the NCAR Command Language, and by 1995 software developers had an early version of the software working. Funding from this work came from the core funds of the University Consortium for Atmospheric Research, the organization that operates NCAR. In 1997, the scientists with NCAR's Climate and Global Dynamics (CGD) division decided to hold a competition to decide which visualization tool they would use for handling data from the Climate System Model (CSM). CSM was changing significantly at the time, adding additional physical components to become a coupled model and beginning to work with outside sponsors and more collaborators. (Soon after this, CSM was rebranded the *Community* Climate System Model (CCSM) to recognize the increased role of scientists outside of NCAR in contributing to the model. In 2010, it was renamed again to the Community *Earth* System Model (CESM) to acknowledge that it models not just climate but earth systems more generally.) One of the challenges in becoming more community-oriented was in sharing the software used at NCAR when working with the CSM. CSM used an old processor for analyzing data and it only worked on the computers at NCAR. It soon became clear that a more portable tool was desirable so that scientists at other institutions could use it with their own computers.

So the 1997 competition was to select a new "official" tool for handling CSM output, suitable for broad community use. The competition criteria included completing as many as possible of ten different tasks that the old processor did already, and involved a series of presentations during one day, at the end of which the CGD scientists voted for their favorite tool. Four tools competed: MATLAB, Yorick, Interactive Data Language (IDL), and NCL. NCL won the winner-takes-all competition by popular vote, with Yorick coming in second. One factor favoring Yorick and NCL was that both are free. IDL had recently raised its licensing fees, forcing the leadership at CGD into decisions about whether to fire staff in order to pay for software. Scientists thus were conscious of the price tag for NCAR, and the obstacle it might present for collaborators at other institutions to afford CGD's tool of choice. In addition to being free, NCL could read all of the major file types that CGD scientists used - not just model output formats but also observational data. The NCL developer who presented the software in the competition also received some coaching from a CGD scientist about what the scientists would like to see demonstrated and would find useful. The final factor favoring NCL was that some CGD scientists liked having the development in-house, believing (correctly) that they would have better access to the developers.

Winning the CGD competition, however, wasn't the final hurdle for NCL's adoption at NCAR, never mind more broadly. After NCL won the most votes in the official competition, the CGD division leader directed one of the CGD scientists to do an in-depth trial period using NCL. The intent of this was to see how difficult it was to learn and whether it actually had all of the capabilities that they needed. A scientist that works with both observational and model data was chosen, to test out NCL as a tool that would serve a broad audience who work with different types of data. This three-month trial period resulted in a few important changes. First, the CGD scientist informed the NCL team that they must change their interface to the graphics functions. The NCL team had produced an object-oriented layer for users to interface with NCAR graphics. The scientist insisted that although an object-oriented architecture may be right from a software standpoint, scientists would find it too complicated and would not bother with it. The NCL team added an interface to hide the object-oriented interface, instead presenting users with graphics functions that work with variables directly, providing easier access to the advanced graphics capabilities of NCAR graphics. The other major observation of the CGD scientist was that NCL only supported rudimentary analytic functions. He offered to contribute his collection of more sophisticated and climate-specific FORTRAN routines, and worked with NCL to incorporate these into the tool. This effectively moved NCL from being a tool that could do file input/output and visualization to one that included a third set of functionality: data analysis. At the end of the three-month period, the scientist agreed that NCL was the correct choice.

CGD had one remaining concern: would NCL be supported and developed indefinitely in the future, or might the computing division cut the product if the budget got tight. To resolve this, CGD met with the Computing division and made a "gentlemen's agreement" that NCL would be supported long-term. By this time, the CGD scientist evaluating NCL had become fairly expert in using it and started training other CGD scientists informally. Soon he was asked to do more and more NCL training.

Initially he did so, but then asked that the NCL team provide someone to help with the training. By about 1999, a new person was hired to help with NCL training, and the NCL team began offering regular classes at NCAR for both staff scientists and various visiting scientists. NCL also began a practice of taking classes to the road, offering classes at different institutions who were affiliated with NCAR. Around 2001, NCL reworked their webpage and began building a detailed library of examples. These examples include code that users can copy as a starting point and then modify for their purposes, with descriptions of what it would accomplish and sample output images when appropriate. Since the early 2000s, NCL has continued on a relatively steady course, adding features, training users, and providing support. Its usage has increased steadily, especially as the large Weather Research and Forecasting Model (WRF) community have become major users.

NCL uses its “ncl-talk” email list for support. Messages sent to the list are distributed to all subscribers, and the NCL team encourages expert users to respond to novices. NCL developers, however, make sure that each message is responded to within a day or two. At times, the support ranges beyond the merely technical, with NCL users or developers advising about the appropriate scientific technique for handling a question about data. Users also sometimes will contribute a script that they have found useful for others to use, and some of the best contributions are polished up and included on the NCL webpage. Other users have written configuration settings to enhance text editors with syntax formatting for NCL; these contributions are distributed through the NCL webpage as well. WRF makes extensive use of NCL, and has a webpage hosting NCL scripts that others can use or adapt to their purposes. Currently the major tasks NCL is working on involve large architectural changes to accommodate the increasingly high-resolution (and hence large) datasets that scientists are using. The NCL team holds weekly meetings as a group of developers, and less frequently will incorporate supervisors at higher-levels within NCAR to get their feedback and perspectives as well. Mostly developers work by themselves on specific tasks, although all the developers are at NCAR’s Mesa Lab and so teammates can easily talk in person as needed. Tasks, including both feature requests and bug reports, are tracked using the JIRA ticketing system.

NCL does not have current user counts, since it does not maintain ongoing contact with all users like ESMF or require user accounts like ESG. Instead, we know that NCL has 1200 users subscribed to its active ncl-talk email list, and 9700 people have downloaded NCL (or its python-based kin, PyNGL and PyNIO) since 2005.

2.3.4 Project comparison

Participants at the 2008 ICSE workshop on Software Engineering for Computational Science and Engineering developed a table to compare different scientific software development projects (Carver 2009a). Table 2 below is an adaptation of the table from that workshop, to summarize the narrative descriptions above and to highlight some of the similarities and differences between ESMF, ESG, and NCL.

Dimension	ESMF	ESG	NCL
Purpose	Simulation	Data sharing	Data processing
Team size	~12	~25	~6
Code distribution	Open source	Not shared	Open source
Project length (years)	10 (2001)	12 (1999)	19 (1992)
Variation among users	Medium	Medium	Large
Software size (lines of code)	~447,000	Not open source	~1,500,000
Processing types	Batch	Interactive	Batch or interactive
Relationship between developers and users	No overlap, but similar backgrounds	No overlap	Some overlap
Governance	Common leadership, multi-institution	Distributed leadership, multi-institution	Common leadership, single institution
Funding	Grants from many sponsors	DOE grants	UCAR core funds
Work place	Many sites, including homes	Many sites, a few people at each site	Single site

Table 2: Context dimensions of scientific software.

The diversity among these projects provides excellent areas for comparison, while involving projects that are of similar scale. The three software systems are in a common domain – all are in climate science and the same scientist may make use of each of them. ESMF has one development team that is distributed across multiples sites. ESG has independent components with more loosely coupled coordination. There is one software system for the web portal, but it relies on software packages handling security and data transport. Different sites lead development of different components of the system. NCL is the most centralized – it is centered at one location and has one specific product. Yet this product resides within a larger set of graphical packages and utilities provided by NCAR.

ESMF spent about six months at the beginning defining requirements, and has since set up other mechanisms for carefully recording and reviewing requirements with a Change Review Board. They use a ticketing system to track tasks, including feature requests and bug reports. ESG uses a ticketing system as well, and writes use cases to generate requirements and then convert them into tasks as tickets. NCL also recently started using a ticketing system to track bug reports and feature requests. Previously they had kept track of tasks through informal to-do lists as text files.

Significantly, the projects have different types of sponsorship. ESG and ESMF are funded through successful proposals to major funding agencies. NCL has a more reliable and long-term funding stream through NCAR's core funds, and promises of continued development to the CGD division. The ESG and ESMF leaders are busy with managing funding and planning future proposals, whereas the main NCL leader is also a developer and is less involved in funding issues. ESMF and ESG engage some of their stakeholders by writing them into their proposals. NCL has a particularly cozy relationship with the users at NCAR's CGD division, but generally does not explicitly partner with a particular group.

The diversity of three different types of software applications, three groups of different people, and three different project histories help validate my findings as general to scientific software rather than indicative of one particular project or type of project.

Chapter 3

Mapping Stakeholders, Counting Users and the Users that Count

This chapter sets out to map the role of stakeholders in my three case studies. A stakeholder is "a person or organization who influences a system's requirements or who is impacted by that system" (Glinz and Wieringa 2007), and we will see how stakeholders are both dependent on the software system, and the software is dependent on them. The term "stakeholder" is deliberately broad, including many different types of people and roles, but my informants refer to people in specific terms and descriptions rather than such broad generalizations. Therefore we will start by defining a basic typology of stakeholders to help map the broad term to specific types of people and the roles that they play within the software projects. We need to first understand the identity of stakeholders, and how and why they become engaged in these projects. The detail provided in this first part of the chapter serves to enrich our understanding of the different types of stakeholders throughout the remainder of the dissertation.

Two related points emerge from this typology of stakeholders, both around the importance of users. First, counting the number of users is an important measure of the success of a project. Ideally, the software projects aspire to provide a positive impact on the production of scientific knowledge. It is difficult and even subjective to measure such an impact, so usually some level of benefit is assumed for each person who uses the software system. The number of users thus approximates the benefit that the software provides to the scientific community. This makes user counts important, and projects respond by counting in ways that maximize the number of users that they have, and by prioritizing adoption of new users. Figures about the number of users are then used to tout the success of the project to sponsors, users, and collaborators, encouraging them to further invest in the software system in their respective ways.

While users counts have an important role, the second point that I have found clearly in the data is that some users play a disproportionately important role in the software project. These powers are experts in using the software, and also develop a relationship with the software team in which they exchange favors. This exchange has great benefits, both to the software team and to the entire user community. At the same time, it reveals that some users are far more important to the emergence of a useful software system, a nuance that is missed in the figures of raw user counts.

3.1 A typology of stakeholders

This brief section develops a typology of stakeholders in scientific software projects, based upon requirements literature and the background information we know about the projects from chapter 2. The typology is intended to serve as a framework to help assess the roles of the many people and organizations that have stakes in scientific software projects. In the early 1990s, those involved in requirements engineering work began turning to the term "stakeholder" in lieu of "user" to acknowledge the interests that developers, testers, managers, and operators (among others) have in the software (Glinz and Wieringa 2007). While users are certainly one type of stakeholder, experts began encouraging people to generalize the concept to include other categories relevant to requirements engineering (Wiegers 2003). So in climate science software projects, the users – typically scientists – were one obvious class of stakeholder, but by no means the only stakeholder.

In the parlance of the people I interviewed, users were often collectively referred to as "the community." ***Users*** may include key individuals who are well known in the community or those who are more marginal, perhaps unfamiliar with all of the knowledge and skills assumed by others. Users depend on the provision of software that assists with the tasks that they need to do. But this is not a one-way dependency. The software development team is simultaneously dependent on users to justify its existence. In the realm of climate science software, users typically do not pay to use the software and so are not directly involved in providing financial support. However, they are critical for justifying sponsorship of the project to funding agencies. Ribes and Finholt have described their role more as a constituency for the project to relate to, emphasizing the sort of political exchange that happens between a software development team and the "community" (Ribes and Finholt 2008). So for participants, to "serve the community" (as they often say) would amount to serving the users whose views are seen as important, and especially those who are politically active. We will look in more detail at different types of power users later in this chapter, including those with strong political influence.

Another category of stakeholder that we see emerging above is the personnel on the software project itself, which I'll refer to as the ***software team***. Software team members have a stake in the work that other team members do, which could affect anything from their daily work life or their career. This is true not just between software developers but also with project managers, software testers, or others involved in running the project. As one project manager put it, the team becomes almost like family since team members spend nearly as much time with each other at work as they do with their biological families away from work. Daily interactions among team members have a significant impact on the work process and ultimately, the software product. *For purposes of clarity, I will refer to those members of the software team who are involved in programming as "developers" and will use "team member" to refer more generically to all people involved in the team whether they do programming or other work.* Also, it is worth noting that in some areas of science an important aspect of the software development process is the overlapping role of users and developers with people who do both science and substantial software development (Carver 2009a). That is generally not the case in ESMF, ESG, and NCL where the software team is focused on building software infrastructure that is not tied to a particular research project. Scientific users may do programming work, but they are generally not modifying the code within the software projects that serve as my case studies. For my three cases, it is fully appropriate to carve out separate categories of users and software team members, even if there is occasional overlap.

In industry, a third category of stakeholder is identified as the customer (in cases of custom software) or the product manager (in cases of a software product). The important thing is that these people have control over funding sources. In science, this corresponds to a third type of stakeholder, ***sponsors*** who provide the funding and support for the software projects. Sponsors have a definite stake in success for the projects that they fund, as they are accountable both to the scientific community and to the public for making good use of public money. The software projects also have a clear stake in sponsorship both for the legitimacy it grants and the vital funding it provides so that they can distribute the software to users for free.

The fourth type of stakeholder is slightly different in nature. Other software projects hold a stake in the success or failure of the software project of interest, either as ***collaborators or competitors***. Such stakeholders could be competitors with an adverse relationship such that one project's success draws users and sponsors away from the other project whose software has similar features. If projects are collaborators with complementary software, then increasing compatibility between two software packages translates the success of one project into benefits for the other too. This is analogous to complements and substitutes in economic theory, which describe goods whose market behavior is coupled. In our case, NCL and ESG are complements in that each makes use of the other, whereas NCL and CDAT are substitutes because the two are competitors that users choose between. Producers of software libraries and software tools also fall under the category of collaborator, but in a weaker sense. Scientific software projects draw upon specialized libraries (e.g., to easily read and write NetCDF files) or use esoteric compilers to build their software for specific supercomputer platforms. They also utilize software produced for more general audiences (e.g., Emacs or Java), but the projects that produce such general-purposes software are only stakeholders in a weak sense of the concept, similar to the role of electricity providers.

This breakdown yields a hub-and-spoke model in which stakeholders are defined by their relationship with the software system of interest. Thus other important relationships are absent from this analysis. For instance, the relationship between NCL's competitors (e.g., CDAT) and scientists/users has a significant impact on NCL, but is absent from this basic typology. Or communication among users may consist of one scientist referring to use ESG to access a certain dataset. This is clearly an important interaction for ESG, even though it does not directly involve the software system. For this reason, software projects may hold a stake not just in individuals but also in processes. Promoting communication among stakeholders can yield benefits for the stakeholders that make the software more valuable, even while the software project team stays out of the loop. NCL does this very thing by having an open "ncl-talk" email list for any user to post questions, and get responses sometimes from the development staff and sometimes from fellow users. This may be one of the reasons that users are often referred to as the "community," as an effort by the software team to cultivate the existence of a community by encouraging users to think of themselves as such.

This typology helps to organize the discussion of stakeholders, as I describe the different stakeholders and how they engage in the software project. The typology is represented visually in Figure 4 below. The arrows in Figure 4 are bi-directional, signaling the mutual dependence that usually exists between the stakeholder and the software system. For example, one might usually say that users have a stake in the software, but it is equally true that the software project has a stake in users due to the importance of adoption for the project's sustainability.

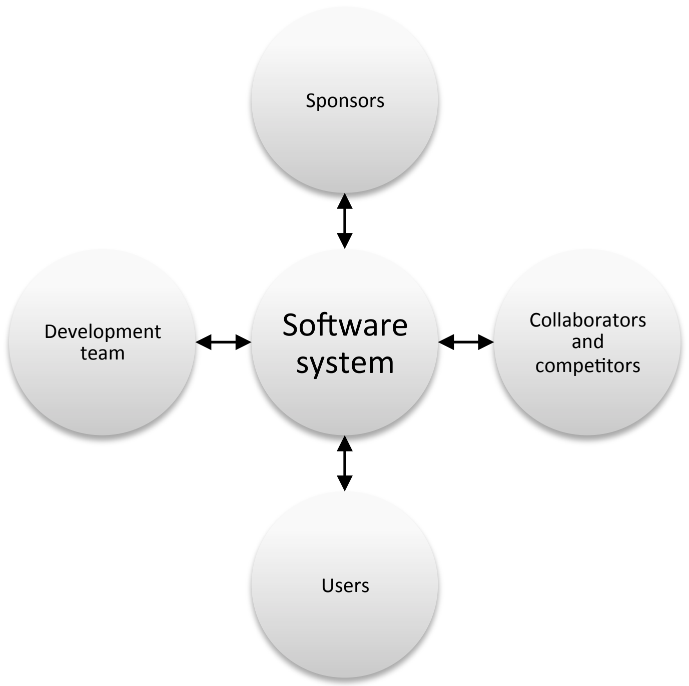


Figure 4: Typology of stakeholders around a software system.

In practice, the relationship of stakeholders to the project are much more rich and complex than even bidirectional arrows. Some relationships are much stronger because of a deep dependency on the software. For instance, users who spend significant energy learning a new system or porting their code to make use of the software have a strong stake in it. We will see later how users are more cautious when adopting software when it takes a lot of work and establishes a close tie. Other times, there is very little investment in becoming a user. ESG is a service that is readily used to download data and does not require the conversion of any user code. After users download the data they need, they are left with little long-term tie to the system due to investment in it. Yet at the same time, ESG is the only source for accessing some data, making users who need those data highly dependent on it. Temporal patterns matter too. Some sponsors may have invested in a project for nearly two decades (such as NCAR with NCL), yielding significant attachment to it compared to a sponsor who just put a few thousand dollars into a special workshop. The combination of these factors affects the language used to describe the relationship a stakeholder has to the software, ranging from "we completely depend on it" to "we are affiliated it" to "we are a potential collaborator."

Groups who establish a financial relationship deepen their stake in multiple ways. Funding supports the time and attention that leads to a longer-term relationship, and is likely accompanied by oversight that leads to a closer coordination between groups. It also often means the groups have decided to collaborate, such that each group is dependent upon the production of the other and hence has a stronger stake in the other group. So it is unsurprising that when funding flows between stakeholders, there is a deeper connection. The following sections flesh out some of this nuance as we consider the role of each type of stakeholder in the ESMF, ESG, and NCL projects.

3.1.1 Software team members

The software team is centered on a group of software developers. However, the software team also includes those who work with the developers on a daily basis: principal investigator(s), project managers, researchers, and personnel involved in software support or quality control. A helpful approximation for who is part of the software team is to note who is paid, in part or in whole, through the software system's funding or who attends weekly "core" team meetings.

Software projects do not arise ex nihilo. Rather preceding events pave the way for the initial idea, funding, and relationships within the software team. The genesis of ESMF was in 1998, when the Common Modeling Infrastructure Working Group started defining what a modeling framework in the climate sciences should do. The timing was also right for NCAR and other laboratories to be receptive to this idea:

> NCAR [was] in the throes of being reviewed by the NSF for its software practices, and it was having a terrible time. It received really a very poor review about the software practices that were going on here at NCAR. It had been criticized for not trying to generalize software as much as was perceived happening at other institutions. It was criticized for using older algorithms, and in general it was just…it was very much in the defensive about these issues when I got here. (Grace #1)

So when NASA issued a call in 2000 for earth system modeling frameworks, the working group participants coordinated to submit three inter-related proposals with high-level support, such as from the director of NCAR. This arrangement of investigators had evolved over the previous years, and then aligned itself to produce a unified project when the opportunity was presented. The participation of GSFC, a NASA center that emphasized data assimilation and the use of NASA data, provided clearer benefits for the sponsors and helped the submission to win the NASA award. Yet at the same time, it was a historical coincidence that GSFC was interested in participating in the proposal. GSFC's Data Assimilation Office and National Centers for Environmental Prediction divisions were just merging, and working out a way to combine their models into a single new model: GEOS-5. This presented an opportunity for significant refactoring of the code and the introduction of a major new framework for the whole model, paving the way for ESMF's first major user. *Similar combinations of serendipity and prior relationships led to the initial team and users for ESG and NCL.*

Building from the original team, the team then must identify and hire suitable new team members to fill out to full size. This can prove to be quite a challenge, as the best developers come with certain rare skills. One developer described the challenge for ESG:

> It's a pretty specific skill set. There aren't a lot of people who…there may be more now… but certainly when I applied and when the funding was renewed, there weren't a lot of people who were doing good computing development. (Tom #1)

Tom had applied to join the ESG team in 2003, and was valued for his knowledge of a particular technology for handling large quantities of data on computer disks. To address this challenge, NCL, ESMF, and ESG have each looked for opportunities to hire internally, selecting developers from other units within their organizations. ESMF has also pursued a tactic of hiring from people who have already used its software. Team leaders have arranged a series of internships that help to build a set of developers who are familiar with ESMF, and could potentially be brought onto the team. At least one intern has done just that. ESMF has also looked to its users as a source of developers, on several occasions hiring advanced users who are already familiar with the basics of the framework and seem to have a set of skills that would be valuable for ESMF development work. An ESMF developer jokingly referred to this tactic as "stealing" or "poaching" from their users to build their team. *Teams build up through targeted hiring of people with specialized skill sets, often looking to known colleagues as potential new hires.*

Team members also have indicated a strong stake in the project because of its ties to their own career and work life. For instance, one developer found himself in a role that involved interaction with scientists and was relieved when this responsibility was removed, preferring to be "shielded" from stakeholders (Lars #1). The role had felt particularly taxing because it involved communicating with scientists regularly, and this proved difficult without more scientific background. Furthermore, the developer's interests were more in programming than in working with the scientific content. The situation was much improved when another team member served as an interpreter of the scientific content, translating it into programming tasks for Lars. The personal interests and backgrounds of team members thus influence the roles that each team member can play, as well as influencing what sort of other roles are needed on the team.

Recent research on the development of scientific cyberinfrastructure has focused on projects that have scientists as both users and developers. This work has found tensions between computer scientists who are more interested in building systems and "domain scientists" who are more interested in research output using the systems (Weedman 1998; Lawrence 2006; Ribes and Finholt 2009). Howison and Herbsleb explained the tension according to the reputation systems that exist for scientists who are engaged in software development, which emphasize publications in the scientist's chosen field (2011). However, this is not the way that it works for team members in the software infrastructure projects we are considering here. Instead, developers are motivated by a career track as a software developer doing interesting technical work, rather than accruing publications. Other team members, like managers, are also more interested in building new systems than in publishing research papers. For some, joining a software team was a deliberate step away from a scientific career (where they had done graduate-level work) into a career that they found to be less stressful, more family friendly, and better paying. *Scientific software projects have thus been able to tap a pool of people who are motivated by financial or other reasons, rather than success in the reputational economy of scientific production, freeing the team to focus on building and supporting the software system rather than being distracted by a priority for producing publications.*

Some, but not all, members of the software team develop a deep love for the software they produce. The software nearly becomes part of their identity, as they are closely associated with the project in the eyes of others and in their own eyes. This result is that some of my informants report keeping their email up most evenings so that they can address project-related issues, or pulling an all-nighter to get some feature into the software that they really wish to finish, or working the weekends to provide support to users. One developer reported apprehension about joining the project, because he expected that he would love doing it and was wary of getting hooked into his work too much. Another developer elaborated on his response to a user's support request that he had worked on the weekend before the interview:

> I probably spent a total of three hours trying to do that. And that is stuff that, I mean it's over the top, totally over the top, but I'm identified with this and I want to make sure that it succeeds. And so, that's why. And in point of fact, even after I retire. I'll probably stay on as a casual and I'll answer questions. (Marina #1)

These especially hard-working team members are important for their dedication to the project and to supporting users, whom they hope will love the software as much as they do. *So some team members have a deep personal investment in the software project and are motivated to participate in the project for reasons beyond just pay or professional reward.*

Team members who are deeply personally identified with the project stand in contrast to team members who participate only for the professional value. Some individuals are identified by the projects as a team member in order to signal the backing of the institution that the individual represents. For instance, the early ESMF proposals had the director of NCAR listed as a member of the team. However, he participated little in the actual running of the project, but primarily served to indicate that NCAR was backing the project. These "proxy stakeholders" represent a stake in the project by the institution with which the individual is affiliated, more than some stake the individual herself has in the project. *So some members may join the team primarily for political reasons to build an allegiance between projects and institutions.*

Software developers are certainly not the only important role on the development team. The level and types of technical engagement with the software can vary. In ESMF and ESG, quality control or support staff often have a responsibility for triaging incoming requests from users and either responding to the question immediately, or assigning it to someone who can handle it. In ESMF's case, this also involves responsibility for testing over the many platforms that they support. ESG is different from ESMF and NCL in that it is a service-oriented platform. End users do not download or see the software, but rather engage with it through a website. So instead, the team must work with system operators who deploy the software that runs ESG on different servers, setting up the federated system of gateways and data nodes that allow ESG to function.

All three projects have one or more people who take on project management responsibilities. Critical functions of these people are to present an outward face to management and sponsors, to secure funding, and to ensure appropriate staffing of the project. Often project managers are the principal investigators who helped initiate the software projects, stabilizing a set of loosely affiliated people interested in the software into a named project by securing funding for the effort. *Project managers have a critical role in the project, especially in providing stability to the team.*

The principle aspects of the project that project managers stabilize are funding and the composition of the software team. They ensure there are sources of funding over an extended period of time so that other stakeholders like users and developers feel free to commit to and work on the project without worrying about its financial health or long-term availability. This is an infrastructural role, making the "project" extend beyond the life of a short-term project into a system that extends across multiple funding cycles and is integrated into many other systems. Ribes and Finholt identify this tension in cyberinfrastructure projects as that of "project vs. facility," arguing that systems that aspire to be infrastructure need the long-term characteristics of a facility (2007). By default, they argue, projects funded by scientific agencies tend to inherit the limitations of short-term funding awards: "this places a finite horizon on long-term activities and shifts efforts from developing stable informational resources to securing post-year five support." My data show that project managers do much of the work that stabilizes a project for the long-term, making it into a "facility."

The importance of project managers became clear when developers talked about funding within ESMF. Within ESMF, Juan expressed his contentment with being unaware of the funding details: "Actually a lot of [the funding work] is transparent to me because Grace just kind of handles it [laughter]. So I don't worry about it too much but think originally I was on a DOD, I don't know, DOD grant, and then I eventually switched to a NOAA grant" (Juan #1). He is unaware of the funding status or details, and liked it that way, preferring not to worry about those aspects of the project. In general, developers were not part of the process of writing proposals, defining high-level project objectives, or obtaining funding. Instead, "Grace" transparently bridged proposals and made sure that the paychecks came steadily even when funding from one sponsor was delayed or by borrowing from other pools of money, and so forth. This represents the main part of Grace's work, and is not at all transparent to her. However, her work on that aspect of the project allows developers to work without having divided attention, and a constant stream of funding enables ESMF to keep developers for a long period of time building their expertise and personal investment in seeing the project succeed.

NCL project managers stabilized the project through the gentleman's agreement with key personnel in the CGD Division at NCAR. Through this, scientist users have been assured that NCL will continue to be developed and supported. In turn, the NCL team can count on a level of support not just from its own division but also from CGD. The NCL team is not worried about funding, especially for the core veteran developers who have been on the project for more than a decade. Although the project shrunk some in the early 2000s, cutting one position, there was no doubt that NCAR would continue to fund development of NCL overall. In a similar way, ESG has been guaranteed continued work and users through its project managers, who arranged for it to be the designated system for distributing CMIP data. This guarantees users, since ESG has a monopoly on distributing some of the most important climate data, and simultaneously makes the software project very attractive for sponsors' funding. As we will later see, user counts and adoption are very important for continued funding, so delivering user commitments to the software does a lot to stabilize the project.

The stability of projects that is created by the work of project managers is extremely important. As one project manager articulated: "I'm actually a person that believes that it takes about ten years to really reach impact, if it's possible to do so; about a decade for large technology projects" (John #2). To achieve this longevity, project managers must work to articulate multiple scientific goals to pursue funding from sponsors mainly fund research. By bridging different scientific efforts, the project managers then include software infrastructure goals as well. Project managers must be adept at translating the value of the infrastructure into its scientific benefits. Rhetoric about the software's growing user base and ample funding also matter, irrespective of the particular status. For projects that take some effort to adopt, such as ESMF and NCL, users want to know that it will be around for a long time before they expend the resources to become a user. Sponsors also want to know that the project is doing well and benefiting many scientists before committing money to it over a long period. If software projects do not really have a significant impact for a decade, then there a lot of optimism is important for helping them to make it that far. *Project managers know that stability of the software project matters so much that they must portray stability as existing until it actually does.*

This section on the software team has portrayed some of the types of people that are on the team, and the different roles that they play. It also emphasizes the motivations that team members have in joining the project. ESMF, ESG, and NCL are well-organized software projects focused on building software using experienced software engineers and project managers. This represents a situation different from that faced by scientific software efforts that involve computer scientists and domain scientists seeking to simultaneously do research and build software.

3.1.2 Users

Users predated the concept of stakeholder as a unit of analysis, and they remain one of the most interesting types of stakeholder. The other types of stakeholders put substantial effort into increasing the quantity and quality of their relationships with users. Quantity and quality of user relationships relate directly to themes of counting (and increasing) the number of users, and the deep relationship that the software team has with certain power users, both of which will be investigated in more depth later in this chapter.

Software team members think of their role as supporting the scientific community, or meeting the community's needs and making work easier for scientists. When saying "community," they have their users, and potential users, in mind. Sponsors also are responsible for serving this community, and so they are consistently interested in hearing about the users that the software is reaching and how it is benefiting users. In the case of ESG and NCL, users are individual scientists who decide to use ESG for accessing data or software, or who choose to use NCL as their tool for post-processing data analysis and visualization. The ESMF user is better conceived of as a modeling group, which ranges from a small group of a few scientists up to consortia of multiple federal agencies. ESMF will typically work most closely with just a few people in the modeling group, but the group as a whole makes decisions about how to adopt ESMF, and the implications of using ESMF affect the group as a whole. ESMF goes about tracking their community very deliberately, using a customer relationship tool called FreeCRM (Free Customer Relationship Management) since 2006: "So anybody that is utilizing ESMF, has been interested in the ESMF, gets put into this database." (Patrick #1) Most of the items in the database are models, with contact information attached for people associated with that model. This helps the team to track current and potential users, and developers are assigned as liaisons to particular users so as to build a relationship with users and ensure regular contact with them. NCL developers also cultivate a close relationship with users, capitalizing with their physical co-location with NCAR to interact regularly face-to-face with CGD scientists who use NCL. For instance, one CGD scientist gives this account of an interaction with the NCL developers:

> So, recently ‘Avi,’ who's one of the main developers actually contacted a number of us who work on CCSM and said, "What are the things that you would like to see come in NCL that would help you?" And so, we actually sat down with him in a meeting and just said here is laundry list or wish list of things of things that we'd really like. And it was really nice because ‘Avi’ could then directly say things to us. (Cameron #1)

NCL developers are also very active on their [ncl-talk] support list, responding within a day to all sorts of inquiries from users. ESG interacts less regularly with users, but still seeks out feedback and offers an email address for questions. The ESG team has used demo teleconference calls as a way of soliciting feedback from a select group of potential users. Because the ESG project focuses on system building over research, joining the software team offers little benefit to the domain scientists who use ESG. Yet the software team still needs their input. Leaders in ESG have thus found it more effective to include scientists by buying their time and attention by funding them on the project:

> …‘Albert’ is our data manager but he is also you know our on the ground advisor about stuff that works and stuff that doesn't. So we fund him directly, rather than asking people to do it out of their good graces (John #1).

In a similar way, ESG has invited scientists to workshops in order to command a continuous portion of their time and energy to solicit feedback on the system. Usability testing during one of these sessions yielded a new understanding that scientists prefer a browsing method for finding data rather than a heavily search-based method. So one way that scientists engage with these software projects is when members of the software teams seek them out for their feedback.

Not all users are sought out equally, or treated equally in their interactions with the software team. Users may become favored either because of a relationship they develop gradually with the software team or because they are strategically important for the software project, perhaps coming from an important institution. The software team splits its attention between supporting existing users, while also facilitating adoption by new users. This may mean helping current users with questions about how to the software, or it could mean working with users to introduce a new feature. For projects that ESMF has partnered closely with, such as the BEI modeling effort, this means tight interaction between developers and users about features: “If we're developing some things for a specific user, we'll interact with them a lot, and we do either support over phone calls or something.” (Juan #2) Support for users who are not part of a partner project is typically done by email. ESMF invites user input during certain “Joint Specification Team” design meetings. Although the meetings are technically open to all, developers have a sense of who the key users are that they want to include in these meetings: “Usually, I mean everything is usually open to the public and anyone who is interested in participating can, but then there are usually these people who in particular we have been talking to with and so we wanted their input on it.” (Juan #1) These few users may have an ongoing partnership as part of a shared grant, or may have built up a relationship with the development team over time. These users are recognized as “friendly users” or “*power users*,” ones who will try out new releases before they are made public and give the developers feedback, or who will offer feedback about a design proposal. They have a great deal of expertise in the software system, and may even create workarounds to fix problems, which developers can reference as they implement more permanent and robust fixes. Some power users contribute code directly to the project, especially those with models that are collaborating closely with ESMF or helping to sponsor ESMF work. These users play such an important role in the project that we will return to them in more detail later in this chapter.

The software teams have also discussed how to handle certain *disfavored users* such as international scientists or foreign militaries from using the software. This is a sensitive issue in part because large amounts of the projects’ funding comes from United States taxpayers through federal agencies, and in the case of ESMF, partially through the Department of Defense. An ESMF team member provided this explanation when asked about what the resulting legal issues are:

> Export laws, for software development. And so there [are] issues that have come and we have not touched them and they have not contacted us since then. That we are not allowed to provide, our server is an open source anywhere in the world can download it but there is export laws that the United States has about who really should be getting your stuff. Do we really want to give stuff even though its open stuff to the Chinese military, in the North Korea, you know all that kind of stuff. So there is these export rules and they were coming around, the original thing that they said

> was because its open source anybody can get it that's fine. But you cannot provide one on one support to anybody from these bad places. (Patrick #2)

ESMF's solution is a pragmatic policy: it's difficult to publicly distribute software without also making it accessible to people from the "bad places." So the team doesn't worry about who downloads the code, but developers are conscious of which users they support and how much time they spend doing so with international users. Given that all of ESMF's support has come from the United States government through taxpayers, the ESMF team concluded that it would continue to support international users in simple ways, but would discuss it if the support work ended up requiring significant effort. The case of international users highlights the influence of the sponsors, even when they are not themselves enforcing prescribed behavior. *Disfavored users may engage with the software system, but in a way that is substantively different from the way other users do.*

Both ESMF and NCL teams also deal with users who ask too many questions through email support, or questions that are about science rather than just the software. As one NCL developer summarized: "some users abuse this mailing list. They ask too many questions" (Melissa #1). Another NCL developer elaborated with a story about a scientist from the University of Reading who asked a question that indicated her confusion was not just about how to use NCL, but about failing to understand her data: "We're a software tool. This is a science issue, and she didn't get it" (Marina #1). In this case, the developer provided extensive help to diagnose the scientific issue, not just the technical one. However, doing so made the developer ponder what level of support is appropriate, and when it is more appropriate to point the user to the scientific issues as the problem. Users are not always clear about what is a technical problem vs. a scientific problem, illustrating the way these can blur together. This tension is only aggravated when dealing with international users, as the NCL developer continued:

> one of the things that we have noticed over the last year-and-a-half is, we're getting a lot of questions from China. I mean, it's free NCL dev support. And the question comes in from my vision point of you is... I know they're processing CCSM data then, there's no doubt. But when they start asking me questions, I'll show you an example of the question that was sent to me by somebody from now, this is not China, but generically it's the same thing from the University of Reading in the UK, is what's the right amount of support? (Marina #1)

One tactic the NCL team uses to resolve this tension is to have its support list be public, rather than just being directed to the development team or a point person. NCL's support list is identified as "ncl-talk," encouraging dialog among users and tapping their expertise rather than simply support from the developers. From the perspective of the software team, users sometimes abuse the support system. *However, from their own perspective, users are simply seeking a way to get their scientific questions answered. When the software support is able to handle scientific as well as technical questions, it is perceived as all the more useful in the eyes of users.*

Even while encouraging users to respond to each other, the NCL software team still answers the majority of inquiries to its support list. In December 2010, NCL users answered 18 of the 57 messages (32%) sent to ncl-talk; the software team handled the rest, and chimed in on some of the former. Users don't just answer other users questions, but they will also sometimes share NCL scripts or utilities that they have developed that they think others may find useful. For instance, one NCL user has generated a configuration file for the emacs text editor that will assist with formatting NCL scripts in a more understandable way. He will periodically send updates/improvements of this script out to ncl-talk, and one of the NCL developers will then put the latest version on the public NCL webpage as well. Through the ncl-talk email list, users have gained a stake in other users, benefiting from the contributions and participation of others. Users are not just a community in the sense of a constituency to which the NCL team must listen, but they are also a community in the sense that they will help support each other. These users come to NCL to help others, and to receive help with scripts for specific tasks that have been developed by and for others.

In addition to the different parts of the software users interact with, there are also indirect, or secondhand users. In the case of ESMF, these are split out as the "application developer" who programs the model to use ESMF and the "end user" who simply seeks to use a model that was implemented using ESMF. So someone who wishes to use the GEOS-5 model must also then use ESMF, involuntarily inheriting the benefits and problems. Similarly, some NCL "end users" do not develop scripts, or only tinker with them in minor ways. For instance, the WRF modeling community has developed a set of

NCL scripts that perform common analyses and generate visualizations frequently desired.

> I think I've mentioned some sites that do real-time weather data that use NCL, and I am not very good at being able to remember everybody, but there's one in Switzerland that uses NCL as their main package for... And these are people that develop websites that have images generated overnight or something like that have images, or interactive graphics. (Avi #1)

Users can run these scripts without much understanding of NCL. Visitors to the webpage are benefactors of NCL, and rightfully count as stakeholders in the project. However, they are a different type of user from those who work on coding scripts. Such *secondhand users who are not directly involved in using the software still have a stake in the software system even though they do not interact with it.*

Finally, software projects face a growing and diversifying user base. As scientists in related disciplines seek to use the software packages, the software team's assumptions about users are less accurate, requiring additional work to adapt both the features of the software and the support process to an expanding set of users. ESG has been especially cognizant of this issue, because of the increased popularity and reach that the team expects as a result of hosting the important CMIP5 dataset. Climate scientist have urged software teams to think about ways to present the data not just to interdisciplinary scientists, but to non-scientists as well (Overpeck et al. 2011). *Expanding the reach of the software system to new users creates new, and to date mostly unknown, challenges in working with users of widely ranging expertise and goals.*

ESMF and NCL are built to support specific computing platforms, and if a specific user's platform is not supported than she will need to do some (and potentially a lot) of extra work to port the software to her platform. ESG avoids this hurdle with its model of server-hosted software that can be accessed through a client on any type of platform. ESMF is distributed as source code, since it needs to be compiled with the user's model code. NCL is distributed as a binary, due to the complexity of the process of building and or porting the software. An NCL developer describes this issue:

> Well, NCL is a difficult package to build in some respects. They make a binary, so, if you're on standard systems, you can just download it and you're off and running. But if you're on a non-standard system, you have to have your HDF libraries, your NetCDF libraries, probably other

libraries, sometimes a software person available so that it hooks up... NCL hooks up with these and then it can process all the different files. (Marina #1)

It is similarly challenging to port ESMF to different platforms, and so the developers do that for users. ESMF has a set of automated tests that run on a nightly basis on (almost) all of the machines that it supports, so testing signals that a given platform (machine + compiler + communication library, in this case) is supported by the team and likely needed by some user. The list of platforms tested by ESMF has steadily increased in the course of the project reaching about 30 platforms as of 2010. Some platforms have also been dropped as computing centers replace older machines and software, but they have been replaced with new platforms. This testing and compatibility with new machines is critical for models. When CCSM planned to run a bunch of experiments on Jaguar at Oak Ridge National Laboratory, it granted ESMF developers permission to use the CCSM allocation for testing. With other machines, the ESMF team purchases an allocation to use for its tests and development. To facilitate adoption for their audience, ESMF developers know that they need to support all the major supercomputer platforms in the United States.

Furthermore, developers install ESMF and NCL onto certain shared supercomputers, so that users may run the software without having to go through the installation process themselves. Users appreciate this, noting that "The fact that the package is free and it's available on just about every system that we use, makes it just the obvious choice for us" (Cameron #1). Platforms that are supported by ESMF or NCL signal user constituencies who are running that hardware and software. Machines on which the developers pre-install the software go one step further, indicating that a particularly important or large group of users utilize the machine. *These levels of support for different platforms indicate not just where the user base is, but also how different users or groups of users are prioritized.*

As we have seen, there is great variety among users and not everyone within the “community” is equal. Some are power users; others are disfavored users. Some are heavy users of the software; others only use it minimally or occasionally. Users use the software in very different ways: compare, for example, the interaction of users who spend a few minutes downloading the data they need from the ESG portal vs. the developer who implements ESMF by rewriting her earth system model. Some end users make use of the software system involuntarily as they access a website or run a model that was created using one of the software systems. The ways and reasons that people become users vary greatly.

Attempts to count the users are thus made more difficult by their great variety. Yet remain important as a success metric to justify continued funding. The general tactic of the development teams has been to cast as wide a net as possible when counting users, and to bootstrap their user base by borrowing or encouraging new users from related systems that already have a set of users. All countable users are important in the sense that they help to legitimize the project and signal its success. At the same time, power users have particularly important roles, and their absence would sorely hurt the success of the software systems. User counts and power users will be addressed more later, where we will detail the ways that *users are important both in their breadth and large numbers and in the depth of their use, which yields power users.*

3.1.3 Sponsors

None of the scientific software projects that I studied charge money for use of the software or support, so they are each reliant on sponsorship by an organization. The projects are costly too. ESMF has cost about $75 million over nine years, largely to pay the salary of the software team. These infrastructure projects have largely hired full-time developers, rather than relying on cheaper graduate students who would also have research goals. With such a vested interest in the project, sponsors are rooting for success and play a role beyond just providing money. As one sponsor puts it: "…from time to time I go to the meetings, and give them some encouragement. And I build this into our strategy and when I go to, to inter-agency coordinations, I bring this out as a highlight of our important activity, and solicit other agencies' participation" (Andre #1). Sponsors take part in publicity, a step toward greater adoption, and help facilitate connections between related projects.

ESMF did not simply win a single $75 million, nine-year award, though! Those funds came from a variety of agencies – DOD, NSF, NASA, and NOAA – and needed to be pieced together one proposal at a time. As mentioned previously, the project manager or principal investigators have a crucial role in bridging grants to ensuring a reliable stream of funding. The project manager works with sponsors to shield the development team from distraction and to stabilize the project over a long time frame. A few years into the ESMF development, team conflict led to a number of developers leaving the software team. Sponsors debated whether to scale back funding to ESMF in light of its questionable status. However, one program manager succeeded in persuading others that this sort of project needs time to work out, and so they should stick with full funding as part of a long-term commitment. The sponsor put ESMF on "probation" but kept the funding levels as they were. Within a few years the project had hired new team members to rebuild the software team, produced important software releases, and found additional partners to help sponsor the project.

A similar long-term commitment was arranged for the NCL project after it won the competition to become the post-processor of choice for the CGD group at NCAR. After NCL won the formal competition, the head of CGD tasked one scientist to learn NCL and determine whether CGD should really accept it as the winner of the competition and commit to NCL. After six months of trying it out and giving feedback to the NCL team, the scientist affirmed that it seemed like a suitable tool on the condition that the team made a few changes, including simplifying the interface for scientists. At the meeting concluding that evaluation period, the head of CGD and the head of the Scientific Computing division (the organizational unit overseeing the NCL team) made a verbal "gentleman's agreement" between the divisions. The essence of this agreement was that:

> Number one, [the Scientific Computing division has] to support, there has to be a commitment on their side to support NCL into the future as far as the eye can see so to speak. It's not going to be 'Oh, in two years, well, we have a budget cut here. We'll cut out NCL. That's not going to work. But if our division [CGD] and if CSM is going to choose it as a tool, it's a long-term commitment… and that they have to support it…and not a nominal [commitment]. (Marina #1)

This informal commitment functioned to give both the NCL team and its new users the confidence that the other will be around. So NCL had a new longevity to the project to begin longer-term efforts and to implement features useful for CGD, without worrying about future financial support or being spurned for another software package. And the CGD scientists could feel comfortable learning NCL, knowing that there will be people around to help them and that the software will continue to develop in future years to meet their evolving scientific needs. Here the division heads worked together to create a stable environment for other stakeholders. The uncertainties of the users highlighted another reason this stability is so important, in addition to the focus and long-term work that it allows developers to pursue. Similarly, the stabilizing work of the project manager and patient program managers worked to instill more confidence in ESMF's fledgling user base. S*ponsors work with project managers to help stabilize the project and create an environment where the software team and users can flourish with fewer uncertainties and concerns.*

Sponsors are able to generate stability for a project, and bring along users both directly and by creating a stable environment attractive to users. This allows the negotiation of an important pair of concerns. Sponsors help signal their "intention of support" while users simultaneously begin to communicate an "intention of usage." This pairing of "intentions" is an important step that must be achieved before the software system can reach a "point of infrastructure" (Pipek and Wulf 2009). As sponsors, software teams, and users begin to agree on the support and usage patterns then the software can become infrastructure with a sustained set of relationships and roles, allowing further iterative "infrastructuring" work that gradually fixes the software system to work effectively for users. NCL's "gentleman's agreement" is an exceptionally clear instance of achieving both the intention of support and the intention of usage, and was a major marker in the maturation of NCL as software infrastructure for post-processing.

Users are also able to help bring sponsors to the software project. In part, what NCL accomplished through the gentleman's agreement was to informally arrange for a certain group of users – CGD – to bring stable funding to the NCL team. ESMF has partnered with users more explicitly, pursuing joint funding with users since the initial grants. The teams of major modeling codes that plan to use ESMF have also partnered with ESMF, helping to fund the ESMF development while they work to adopt ESMF. This raises the question of what their role will be in setting ESMF's priorities and direction, an issue that the ESMF team has tried to handle directly:

> Now there's a lot of new people and they're actually financing this thing. And we did have a lot of discussions about okay, when you have some group of people who are coming with dollars and others who are not. And

> then, let's say the group with the dollars wants this feature done first and the group with other dollars want another feature done first. How is the change review board going to prioritize these requests? (Jennifer #1)

As a consequence of the move from NASA's funding of core development in the first years of the project to a multi-institutional funding model, the Change Review Board was formed to manage competing development priorities. A NASA person noted this change:

> Well, as external users start making changes and requesting certain kind of functions, and their funding, their voice started to be heard and the change control board started to have... You can notice the different agencies role has changed, has evolved. NASA's role at the beginning was very strong and NASA's role has weakened, and other agencies have picked up that role.
>
> …
>
> [The project manager] and the Core Team has still a lot to say [about] what gets in there; the control board for example. It's a painful process to negotiate something that would get into the ESMF core software, right? Of course, a lot of the discussion is on priority. Now, once you start dealing with priority, you start [saying]: okay, NASA wants A and Navy wants B. Of course, Navy put in more money now, and Navy's interests get taken care of first. And NASA people are not happy and that became a backlog. (Andre #1)

So changing sponsors changes the direction of the project, and as the core development wrapped up, the ESMF development efforts moved toward supporting the needs of the modeling applications that are sponsoring it and partnering closely with it. The ESMF leaders are active in watching requests that come into the esmf_support email list, making sure that developers understand when the request is coming from an important partner so that they can respond thoroughly and promptly. This may even involve developers drafting emails for circulation within the group to ensure clear and accurate communication. Of course, people from those sponsoring models also have an interest in tracking the use of their money, so they pay more careful attention to ESMF's work and have more opportunity to assert their preferences. *So one reason for sponsoring ESMF is to gain more influence in setting development priorities that will meet the needs of the sponsoring organization, or the needs of the particular users who help arrange for funding from a sponsor.*

Instead of major partnerships, the ESG team has engaged in a much more mild – yet still important – form of relying on funding from the modeling centers for whom it hosts data. ESG is dependent on resources given by the model centers that agree to host a data node to distribute their model data:

> These resources are not provided for through this project's funding, so we are reliant upon the capabilities provided by our collaborating centers. For example, LLNL, ORNL, LANL, LBNL, and NCAR spend core funds on large storage systems to store and serve ESG data holdings. The inclusion of these and other external resources may seem blurred, but the loss of any one of these crucial resources could seriously jeopardize the success of the ESG-CET. (2006 ESG-CET Project Management Plan)

So outside of formal financial contracts, the informal partnership with centers that host data nodes amounts to the exchange of financial benefits. In addition, these centers all have ESG Co-PIs, so funding goes to portions of the center to do development work, and then, in effect, informal benefits are given back by other portions of the center via core funds that support hosting a data node. Like with more formal sponsorship of software projects, this informal sponsorship is provided by organizations whose mission includes providing benefit to the users of the software. *Sponsors become involved by helping to stabilize the project financially, and connect the software team with users and provide a longer-term basis for the project.*

3.1.4 Collaborators and competitors

In addition to assembling its own team, the software team collaborates with some projects and competes with others. These types of connections between projects are numerous, both among the three projects we are considering here and with many other scientific software systems. ESMF has collaborated with the ESG project to help generate metadata that are compatible with ESG. Models that are run using ESMF will soon automatically generate a set of metadata describing each model that is run, so that the process of configuring a model to run an experiment would also generate the matching metadata. ESG can then more easily describe the data it is distributing by using information about the models that were used to generate the data. As of summer 2010, ESMF is preparing this "attribute" functionality for the CMIP5 experiments to be run with CCSM.

Integration into ESG has required substantial work by the ESMF developers. In another case, ESMF has become linked to the Climate-in-a-Box project with minimal effort on the ESMF team's part. The Climate-in-a-Box is a Cray computer for climate scientists, which comes pre-loaded with a number of models and other tools needed for climate science work. Managers at NASA who conceived and now oversee the Climate-in-a-Box project were aware of ESMF and so included it as part of the default configuration of the computers, supporting climate and weather models such as GEOS-5. *Collaboration between projects sometimes involves much communication and close interdependencies between technical components, but other times the collaboration can be much more loosely coupled.*

Software projects also need to coordinate with the teams that produce core software libraries and file formats. There are a number of libraries and formats that are specific to scientific work, and one of the advantages of working with a tool like NCL is that it can handle many different formats. In order to provide this transparently easy functionality for its users, NCL developers must work to ensure that they keep up-to-date with formats and handle them correctly. An early selling point for NCL was that it handled GRIdded Binary (GRIB) data. A CGD scientist recounts:

> I was already partial to NCL, although I had never used it, because it handled GRIB data. GRIB is a format used by the operational weather centers, and it is an ugly format. It's a record format, not even a file format. And all I'm going to say is that it's a nuisance to deal with, and especially back then. And NCL handled it. And what NCL does is, I think, from a person using that language…whatever data it imports, it makes the variables look like they came off a NetCDF file. So that means if you're dealing with GRIB, it imports all of the information, and it has some built-in software that looks up these GRIB tables that are produced…and there are many of them, many, many of them…produced by the operational centers. It goes in and grabs all of this ancillary information called metadata, like what the name of the variable it is, what the units are, what coordinates they're associated with, and it imports this variable. … So you bring it in in, one line, and it's all there. (Marina #1).

You can hear the tone of relief to have software handle the onerous parts of her job. NCL handles the hard part of keeping up-to-date with the GRIB format, moving it from a task for scientists to a task for the software team, and making it easier for scientists to just work with the data. NCL can import a number of different file formats, presenting each as a simple object within the NCL interface. Currently NetCDF is the most important file format that NCL handles, and so here a developer recounts the work the NCL team must do to coordinate with a release from the NetCDF group to keep it simple for scientists:

> For example NetCDF, they are on a new release... As you can see, if NCL wants to use the feature of the NetCDF new release, they have to wait until after [the NetCDF team] releases. Then we have to change our code because NCL uses the NetCDF library to handle NetCDF data. So you have to wait until NetCDF releases their code. And then we'll have this feature and then we can change our code accordingly. (Melissa #1)

NCL will seek to time its releases to come soon after a NetCDF release, so that it can relay the benefits of the updates to the file format down to the users of NCL. The benefits run both ways. NCL was recently working on updating its support to include the latest version of Hierarchical Data Format (HDF) files. The NCL developers benefit by having a defined file format and libraries available from the HDF team, and the HDF team benefits from having NCL support their file format. In fact, HDF team members asked the NCL developers to provide for them some basic scripts that illustrate how to work with HDF files using NCL scripts, so that they can post those scripts on their website and advertise that there are easy ways to work with HDF files. There is clear and immediate benefit to both software projects through this collaboration

While some collaborative efforts involve integrating two software packages, others focus on working out and standardizing data formats. ESG has spent a lot of effort providing feedback to the Metafor team about the metadata standard that Metafor is developing to describe model configurations. This "data modeling" involves "working closely with our European Metafor partners to figuring out how to best describe, or figuring out how they've described the metadata and communicating that to us and making sure that that model is represented in our infrastructure. So, a lot of it has been collaboration with the Metafor team." (Lars #2) The ESG and Metafor systems are loosely coupled, with ESG downloading new data that is posted by the Metafor team. So deciding on a standard format that works for both parties is important, but the software code does not need to integrate. The collaboration also provides a compelling case for Metafor as it asks modelers to fill out a survey for them, since the metadata collected from that survey is then be passed to the ESG and distributed with CMIP5 data as part of

IPCC AR5. This collaboration to develop metadata standards has proved quite challenging, showing that collaborating on standards is not necessarily any simpler than the collaboration that occurs when a software library is tightly integrated into another software system. *Both forms of collaboration are attractive to software teams who can benefit by increased exposure, gaining users and becoming important for the functionality of another system. These effects bolster the software's legitimacy and prospects for future funding and influence within the user community.*

NCL's relationship with ESG shows how collaboration can also lead to competition. ESG has recently been trying to vertically integrate data access, becoming a site for executing a scientific workflow from model configuration to raw data to processed data and into statistics and visualizations. In doing so, it ends up collaborating with many other software projects. OPeNDAP (the Open-source Project for a Network Data Access Protocol), for instance, is used to remotely access subsets of data, reducing the data that must be transferred over networks and transferring just what is needed for the current analysis. ESG thus collaborates with the OPeNDAP team, even to the level of sharing funding with them and listing them as part of ESG funding proposals. For some components, such as OPeNDAP, ESG's choice of a tool is relatively straightforward since there are no strong alternative software systems. For others components, like a post-processing tool, the choice is not obvious. Choosing a post-processing tool to generate statistics and visualizations is particularly sensitive because the ESG Principal Investigator (PI) at NCAR also oversees NCL, and the ESG PI at PCMDI also oversees CDAT. NCL and CDAT are rival software packages that do essentially the same things, and each team would identify the other as a competitor. Both have an interest in being the designated tool in ESG because that leads to more users, greater impact on the science, and increase publicity – all things that help lead to further funding and success as a project. The ESG team's solution so far has been a compromise. Both NCAR and PCMDI deploy ESG gateways that separately catalog and access any data from the data nodes within the federation of ESG partners. So the plan is to use CDAT as the post-processor on the PCMDI gateway and NCL as the post-processor on the NCAR gateway. This will incur greater costs in terms of integrating the ESG software with both CDAT and NCL, but succeeds in keeping different stakeholders within the ESG software team happy.

ESMF's relationship with CCSM illustrates another case in which software projects compete for a valuable collaboration. CCSM has partially adopted ESMF, and uses ESMF's infrastructure functionality for time keeping (i.e., to keep track of the date and time of the model run). However, users can choose whether to use ESMF interfaces between components or to use interfaces based on the Model Coupling Toolkit (MCT). MCT is developed at Argonne National Laboratories, and is intended to be a lightweight package for coupling model components. MCT has a simple datatype that can be used to exchange data. MCT has been used in CCSM since CCSM3 was released in 2004, before ESMF was ready to be used widely. So MCT provides a basic interface that works for coupling CCSM components. However, ESMF supporters have argued that using standard ESMF interfaces would make it easier to couple with components from other models. In the latest releases – CCSM4 and CESM1 (CESM, the Community Earth System Model, is the successor to CCSM) – both MCT and ESMF options are possible, with MCT enabled in the configuration by default. Scientists, or at least those at NCAR, see relatively little benefit to replacing MCT: "[MCT] is a nice lightweight package that works very well for our purposes" (Cameron #1).

With scientists' interest in lightweight and fast tools that will allow more model computations to be completed, much of the debate has revolved around the relative speed of the model when run with MCT as compared to when it is run with ESMF. ESMF has had one developer spending time to get ESMF working and configured within CCSM, and has also produced a set of performance comparisons that the developer presented at the CCSM annual conference in July 2010. Before presenting the findings, developers from the two teams met to discuss the results and make sure each believed his product was being fairly represented. So while there is certainly a sense of competition between the two software systems seeking to fulfill the same function, there is also a collegial relationship between them. The sense of competition is fueled during times when potential funding or users are at stake. As mentioned in the discussion of sponsors, NCL participated in an uncommonly pure, top-down form of competition. In that case, an explicit set of presentations and voting by the audience of scientists led to substantial funds and users becoming dedicated to the NCL software package. While the NCL event was directly framed as a competition, we see from the experience of ESG and ESMF that scientific software packages are facing other forms of competition all the time. Competitors seek to define the relative advantages of their software to users, and to position their software system for long-term success in adoption and funding. Yet among those working on scientific software, individuals also see their competitors as colleagues. *Software team members may simultaneously collaborate on one project while competing on other projects.*

3.2 Counting users as a metric for software success

Commercial software teams use revenue to measure the aggregate value that users place in their software products. Under a given price scheme, revenue thus corresponds to some number of users. Revenue is readily determined, and provides an immediate metric for how the software team is doing. Teams that offer their software for free do not have a similar measure of revenue, and have to approximate their users through measurements of the number of registered users, or of the number of downloads.

For some open source software projects, precise measurements of the number of users is not important because the software developers volunteer to work on the project out of their own interest rather than a need to reach large numbers of people. There is thus no need to justify the efforts to funders. Larger open source software projects such as Linux or Firefox do need to measure user acceptance, since development is partially funded by foundations and corporations. However, these projects are sufficiently large that use of the system is measured through large surveys of computer users to determine market share.

No such surveys exist to measure the use of software among climate scientists, and scientific software that is provided for free also lacks any revenue results that would indicate how it is being accepted. Yet the software teams still need to justify their costs to funding agencies by demonstrating the aggregate value of its efforts. It is difficult and controversial to try to measure the benefits to research that may accrue by using a software system, so instead the primary metric that software teams use consists of various forms of user counts. Such figures are reported in presentations and reports, implying that sponsors can then assess the total value the software system is providing to the community as they consider how they should fund it.

In this way, sponsors provide a sort of indirect revenue that is based upon approximations of user counts and approximations of the value that users place in the software. Potential users also consider the popularity of a software system to help them assess the likelihood that the project will have a community of fellow users and will be around for a long time. *So having large and growing users counts encourages sponsors and users to invest more in the software system.*

3.2.1 Adoption increases users

Given the importance of counting users for scientific software projects, we would also expect an emphasis on getting new users to adopt the software system, and this is just what I have observed. Here is a brief tour of the ways that each of the three projects promote adoption and gain users. ESG has the most straightforward path toward getting users to adopt it: it has been designated by the IPCC as *the* way for disseminating the data for models that participate in CMIP. When the IPCC asked PCMDI how it planned to distribute CMIP data, PCMDI leaders simply chose ESG as the method, since PCMDI was also part of the software team. Scientists who wish to participate in CMIP are thus forced to become ESG users as they provide datasets to be ingested into the ESG and distributed for download. These people cannot just do this through the ESG website, but instead the ESG team helps them to transfer their data with all the correct metadata. Datasets are then checked after they are delivered to ESG, validating their conformance to metadata standards like the CF conventions. A list of the modeling groups participating in CMIP, and hence contributing data to ESG, looks like a "who's who" of many of the best climate science research centers in the world, and their adoption of ESG immediately lends credence to it.

Scientists who wish to use data that is hosted on ESG must also visit an ESG gateway website and register as a "user." They are then granted permission to download certain sets of files. All of their downloads are tracked so that ESG is able to generate detailed usage metrics, which data providers find helpful. Using records of how much their data are downloaded, modeling groups who uploaded data can report to funders about the use and influence of their model data, supporting their petitions for additional funding. *Data usage works for modelers in a way similar to that of software usage for software teams, justifying further funding of the project.*

While scientists with modeling groups appreciated having their data distributed, they did not typically use ESG to access data for themselves. This became evident as one NCAR scientist explained about how to use ESG: “Well, I think you browse through the menus and so forth. Quite honestly, I don't use the ESG myself” (Cameron #1). This individual was certainly registered on ESG as a “user” and benefits from the data usage metrics recorded by ESG and the fact that other scientists can get model data through ESG rather than bothering him. However, he did not use the ESG system to get data for his own work. NCAR scientists have local access to machines with copies of the data, and understand the directory structures used to store those data. They also are fully familiar with the way that the data are produced, leaving little reason to use ESG to access the metadata about datasets they need either. Similarly, scientists who are familiar with NCAR data but are not based at NCAR may bypass ESG and access the data directly. Even with technological improvements in transferring large quantities of data, the size of climate datasets makes transfer over a network tediously slow. So scientists sometimes prefer flying to NCAR to directly access the local storage systems, transferring the data they need to portable hard drives. But if a scientist needs data from another model, or a scientist does not have direct access to model data, then ESG is the primary way to get it.

Because ESG is the designated system for distributing CMIP data for the IPCC, it gains publicity and use more broadly outside of the community of climate scientists. This growing breadth of adoption is a concern for ESG’s managers. One noted that in CMIP3 – which also used ESG – there was relatively little confusion about the metadata categories since scientists were the primary users, and the software used the CF conventions that were familiar to scientists. However, as the IPCC process gains political attention, ESG developers anticipate a much broader set of people using CMIP data, such as politicians, hobbyists, working groups 2 and 3 of the IPCC (responsible for assessing impacts and mitigation), and perhaps even high school students. The diversity of this expanding user base will lead to people with all different types of scientific backgrounds trying to use the data, so the ESG team's assumptions about the sufficiency of CF conventions will no longer hold.

The expanding base of users has been a challenge for developers who are designing the system. During one teleconference call in which developers were demonstrating the latest version of ESG, one developer wondered, "Who are the customers? The answer depends on who you're providing information to. Anyone who will use ESG, not just IPCC. Certainly those downloading data for the IPCC assessment" (ESC meeting). Developers still have a sense that their core audience is scientists involved in the IPCC assessment process, but they are also conscious of wanting to support a much broader audience. ESG's use cases, intended to drive the design process, have targeted the needs of IPCC scientists. A broader set of users is helpful in justifying the importance of the project, but adds a significant challenge in designing for a new and distinct set of user needs.

NCL does not have a monopoly like ESG, but does receive preferential treatment within CGD at NCAR. In 1997, CGD held a competition to determine which post-processing program should be designated as its primary tool (and hence become the primary tool to be used with CCSM data). NCL, developed within NCAR, beat out the competitors from outside, including MATLAB, IDL, and GrADS. The institutional and physical closeness of users and the software team contributed to this match, improving NCL's chances during the competition and then facilitating adoption by individual scientists who work at NCAR ever since. NCL developers have great access to some of their users, allowing them to prepare a winning presentation during the competition. As one CGD scientist recalls:

> The person who presented NCL was a guy by the name of Ethan Alpert, and he really…he did the best job. He did talk to me prior to doing that, and said: "What is it that you think it is that people want to see." And I said: "well, I am a data person that occasionally looks at model output, and these are the things that I would like to see. (Marina #1)

Today, scientists continue to appreciate that NCL is developed locally, and value the access this grants them to the developer team. As a CGD scientist summarized in 2010: "it's developed here, the developers are here locally. It's tailored particularly to the community models that are developed here at NCAR" (Cameron #1). *The "home court"* *advantage of personal connections and interactions that allowed NCL to be successful in the competition just foreshadowed the ongoing benefits that scientists identify from having access to the developers.*

After NCL became the post-processing tool of choice within CGD, its usage steadily grew. NCL scripts were packaged with CCSM, and so the many CCSM users worldwide were exposed to NCL and benefited from using the same post-processing tool as their peers at CGD. Later, NCL became a favorite tool within the WRF modeling community (also based out of NCAR) and a number of scripts were developed for working with WRF data. *NCL bootstrapped its adoption by being bundled with software that already has an extensive base of users.*

NCL also took advantage of being the tool of choice at NCAR. NCAR is a hub of the climate science community, and as scientists visited they learned NCL before returning to their home institutions. NCAR hosts many graduate students and postdocs who then learn NCL and take it to their future employers, including universities where they help train more users. The effect of educators on adoption is important, as one university professor explains:

> A: The thing that I like about NCL is that it's free. So that students, no matter where they go, can use it. They can download it. They can put it on their laptop. They can use it from anywhere. And that, to me, is pretty valuable, because if you learn one plotting software and then you go, you finish your degree, you go to some job, and you have to buy some site license for IDL, I just can't. ... I started using NCL because it makes the best publication-ready figures. I think it has the best graphics. … I started using it in 2005. Before that I used another software package called GrADS, which is also free, but has really poor-quality figures, and I got tired of having really crappy-looking figures that I send for review, so I switched to NCL. It was actually really hard to do, because once you start using a software program you start writing a bunch of scripts, and then you're just always reusing scripts to do the same thing, so it was actually a very painful transition to do. But I'm really glad I did it. I think it paid off. I suggest my students…if my students don't know any other script I support it, I tell them use that. I tell them to use NCL. I had a postdoc last year who insisted on using IDL, and I didn't make a big deal about it, and that was a mistake. Because then he left and didn't finish his figures, and I had to kind of botch through IDL to get the plots made. So I'm never going to do that again. I'm going to make sure everyone uses the same plotting software. Because it's really important for continuity, and things

like that. So in the lab [for a course she's teaching] we use NCL. All the students have learned how to use NCL. (Ashley #1)

This account of adopting NCL illustrates the mixed motivations of users, identifying both technical and social reasons for adoption. She encourages her students and postdocs to use NCL. For students, she feels like it is better to teach them a tool that they will be able to access in the future, protecting them from a difficult switch like that which she went through. For postdocs or graduate students who do research with her using NCL is important for a different reason. Scientists using NCL build up a "personal software infrastructure" (Howison 2011) of scripts that they use to do common tasks or to generate graphics that they anticipate using in publications. Producing this collection of scripts represents both a substantial investment, and yields significant savings in work over the years. In practice, this is not just *personal* software infrastructure, but infrastructure shared with close collaborators. Climate science researchers regularly work with colleagues, so the scripts produced by one member of a research team are used and yield benefits to others in the team. However, if only one person understands the scripts, and then departs, it becomes quite difficult to use them. This collaborative use of scripts is important for adoption, because it serves as a way of propagating choices about a scripting language, depending on who has skills or power in a given situation. When junior scientists work with a mentor or visit an institution like NCAR, there is substantial incentive for the senior scientists to ensure that all the work that is done uses NCL so that they will be able to understand and use it later. Junior scientists then begin to build up their own collection of NCL scripts, the semi-personal software infrastructure that they will bring with them to their future jobs and research collaborations. *Software systems like NCL that are important to scientists' personal software infrastructure makes for dedicated users and provides a mode for the gaining new users through research collaborations with existing users.*

The technical features of NCL are also very important. In fact, it was the high-quality graphics from NCL that pushed Ashley into switching to it. This factor ties directly into the academic reputation system, as you can hear her concern about how reviewers will perceive her work. In addition, NCL is just easier for her to use than some of the alternatives. It supports all the grids she needs to use, saving time and frustration so that she can devote her energy to things that matter more to her than fiddling with technology. The ease of use for tasks that she is doing matters for students too, as it enables her to ask more of students if the tool will require less from them.

ESMF has a more difficult task in persuading people to become users, lacking ESG's monopoly and NCL's spread through personal software infrastructure. Instead ESMF looks for ways to facilitate the adoption of ESMF by new users, be it by supporting them in the installation process or adding features that they require in order for ESMF to meet their needs. For users, adopting ESMF takes time and money, sometimes with no immediate benefit in return. The decision to adopt is usually made at the level of the organization since the model is shared software, which requires achieving some level of consensus.

Adopting ESMF has been more popular at management levels than it has been among developers. One modeler puts its "magical" draw like this:

> … they can really leverage all the technology and doing there and this middleware is just going to make that thing magically work together. They want it because it's cost-saving; it's leveraging. It's interagency cooperation. So if you think about NOAA, it percolates to very high levels. (Rahul #1)

However, model developers who face the work of converting a code to ESMF can determine how they will implement this high-level requirement, sometimes adopting for political reasons only: "And you say, 'OK. They want it; I'll give it to them.' Okay? They hire someone and say, 'OK. Go learn that thing called ESMF and see, but just keep it to a minimum.'" (Rahul #1) One common response by model developers is to selectively adopt parts of the framework, but not the whole thing. In this way they can extract the parts that they see most valuable, and avoid aspects that seem onerous, while satisfying the organization's decision to start "using" ESMF. A scientist with one of the major modeling codes explained:

> Well so, as I said earlier, there are pieces of the ESMF that are very useful. And so that's the way we've intended to approach it is making pieces and using them instead of using the overall overarching framework itself. (Cameron #1)

Partially as a result of this selective adoption, the list of model components using ESMF has grown quickly, reaching some 90 components as of 2008 (Earth System

Modeling Framework 2009). The problem, both for the ESMF team and for outsiders, is that these components have adopted different aspects of ESMF and so it is hard to tell what it means to be an "ESMF component." For instance, a model could use ESMF for regridding or for its time management system, but not use the superstructure that allows ESMF components to couple with other ESMF components. This runs the risk of undermining one of the original purposes of ESMF: facilitating coupling between different components. Consequently, the ESMF team has developed a guide to different levels of adoption, and has implemented a compliance checker to automatically check a code's ESMF adoption. Building a compliance checker was a political exercise as well as a technical one, as the team is conscious of the risk of alienating its major users and downplaying their work, if they were to designate the user's code as non-compliant.

Early on, in an effort to promote interoperability between model components, the ESMF team decided to require that users make their software conform to its standardized framework. The obstacle of conforming to the ESMF standard is compounded by the asymmetrical stakes that the models and ESMF have in adoption. As Leroy adds: "NASA, MIT, NOAA… so they all have their own agenda, but they all work on different unrelated models. And they use a service that NCAR or others provide for ESMF. And so ESMF is [in a bit of an] artificial situation. It's not an essential route for modeling development, modeling activity in the country." (Leroy #1) Major modeling codes that are considering converting to ESMF are unlikely to reap substantial benefits. Instead, their adoption benefits the ESMF effort, which adds to its user base, and benefits other modeling groups, which could then more easily link their components to the newly ESMF-ized ones. However, major modeling codes have plenty of scientific work to do within their own model, without bothering to connect to others.

Consequently, the largest and most thorough adopters of ESMF have been either new models or ones that are planning a major refactor of the code. In its first phase of funding, ESMF partnered with Goddard Space Flight Center at an opportune moment: "So Goddard fully embraced the ESMF and a couple of advances I noticed was they were developing new model and they need this big capability, so they fully embraced ESMF" (Andre #1). Similarly, as ESMF transitioned to its second phase of funding, BEI and then NUOPC both opted to use ESMF as they started to think through their efforts to create new coupled models: "[BEI] had just come up on this idea that they were going to need a framework because they were going to couple models that they had not coupled before. So, instead of writing their own, they decided to throw their weight behind the ESMF. So, that was a whole new community that came into the middle of the project" (Jennifer #1). Major models wait until they are doing a large reorganization of code to integrate ESMF, and then they partner with the ESMF team at a level that involves sponsoring some ESMF development work. For ESMF, this partnership involves extending its functionality with features needed for this particular model and aiding the modelers with integration of ESMF into the model code. Similarly, as ESMF has considered how to help CCSM to adopt ESMF, it has at times assigned a developer to work on the CCSM code and to help them adopt ESMF. It takes a lot of effort to adopt ESMF, such that modeling groups will not usually choose to afford it on their own.

Another strategy to increase adoption is to use publications and presentations at conferences and workshops to raise awareness of ESMF's capabilities. Such rounds of publicity are a deliberate part of ESMF's strategic plan, particularly during key moments in the project's progress. For instance, upon the first release of ESMF with stable interfaces at the end of 2010, the team's goal is to:

> Attract new users and create a positive impression of ESMF in the broader community. The ESMF project had an intense period of publicity at its inception, and another with the delivery of demonstration applications at the end of the first funding cycle. The ESMF team plans one more round of publicity to coincide with the delivery of a functionally complete package and a paper reviewing the transformational impact of ESMF, technically and socially, across the modeling community. (Earth System Modeling Framework 2008)

ESMF developers presented about their project at the American Geophysical Union fall meeting and at the Earth Model Coupling Workshop, both in December 2010, to highlight ESMF's new features. The aim is to introduce people to the software so that individuals and modeling groups who had decided against using ESMF in the past might reconsider in light of its new capabilities.

ESG, NCL, and ESMF face different challenges in seeking to gain users, and helping people to adopt their software system. When adoption has been difficult, such as with ESMF, some users respond by partially and selectively adopting the system. When adoption is relatively simple, as with ESG, individuals may use it without much thought. Such users do not have nearly the same allegiance and investment in a software system as do scientists who develop a personal software infrastructure based upon a system like NCL. *Adoption means different things not only among the users of a single software system but also when comparing across systems.*

3.2.2 Who has adopted? Who counts as a user?

Software projects know it is important to increase user adoption, but they also need to document that adoption by counting users. This is a tricky process. All three teams have put in place systems to help track the numbers of users they have. ESMF does this with the FreeCRM tool mentioned previously, and regularly uses a cluttered graphic listing all its users to help communicate the breadth of modeling groups who are using ESMF. This information is included on its website, in public presentations, and annual reports to show its progress and persuade others that it is gathering momentum or may have even achieved critical mass. NCL and ESG similarly use a combination of visualizations and statistics to communicate that they have a large and/or growing user base.

ESMF and NCL both consist of software packages that may be downloaded by users, and the teams seek to track these downloads carefully. Downloading ESMF involves filling out a brief form, and then following a link to a file with the source code. To download NCL, one must create an ESG account and use the ESG gateway to obtain the file. ESG had developed a system for closely tracking file downloads. This was originally intended to track the use of model data, but the NCL team appreciates the way that it can help them track downloads based upon a user account rather than merely knowing that the file was downloaded to a given IP address. It also provides a little bit of data about users, such as their country and institutional affiliations from a form filled out when users initially register with ESG. Symbiotically, users who download NCL from ESG also benefit ESG by increasing the number of ESG users and showing users the system for accessing data that they may be interested in. This tracking is attractive to other software projects, and CESM also distributes its model code through ESG in order to take advantage of this tracking. As ESG team members reported: “Between June 2004 and May 2010, there were roughly 11,500 users of the NCAR ESG portal,” and “of that

11,500, about 9,700 downloaded NCL, PyNGL or PyNIO." A majority of ESG's users were actually mainly "there" in order to get NCL. During this same period, there were ~1,300 downloads of CCSM/CESM model code and a similar number of users (~1,200) downloaded CCSM2/CCSM3 data. It is clear that ESG is sharing users with NCL/CESM, and that this partnership offers tremendous benefits in terms of the user statistics that ESG can report.

Of those who used ESG to download data, some 200 TB of data were downloaded. A mere 21 users accounted for about half the total volume download, averaging ~10 TB downloaded each, while the full CMIP3 data collection is 35 TB. The top 56 users accounted for 75% of data downloads. As in many other domains, such as music consumption, data consumption is not uniform (Anderson 2008). Some users download a disproportionately high quantity data, and ESG also caches certain datasets to speed access to them because some data are much more commonly requested than others. This highlights the heterogeneity among ESG users, yet annual reports that include user counts do not discriminate users who download software code from users who download just one dataset from users who download large quantities of data (Earth System Grid 2006).

ESMF uses a simple form (similar to ESG's) for users to complete when downloading the software, requesting information such as name, email, country, and institution with an optional field for "Scientific or Technical Interest." For both ESMF and NCL these download counts are subject to biases. They count downloads, rather than use. A single download could be installed on multiple computers, and some downloads are installed on a shared supercomputer which many people use to gain access to the software. Some of ESMF's most important users do not need to download the software at all, since the ESMF team makes sure that correct version(s) of the software are installed on the supercomputers used by those users. On the other hand, download counts can also over-represent users, such as in cases where users download several different versions of the software, download the software to multiple computers they own, or download the software without ever successfully using it. Software teams can sort through some of the errors in download counts by trying to track downloads by registered users, but even the best tracking cannot account very well for the range of types of users illustrated by the ESG example above.

The emphasis on tracking users and reporting user adoption highlights its importance for securing ongoing and new sources of funding. Funders echo this as a priority, replying that when ESMF is successful, "You will start to see more adoption" (Andre #1), and particularly when core models are ESMF compliant, achieving a greater degree of adoption. ESG utilizes user metrics along with IPCC endorsements to emphasize the importance of continued funding. Even NCL, which is on UCAR core funds rather than grant-based funding, carefully tracks and reports on its growing user base. While we sometimes emphasize how users have a stake having good software systems to support their work, it is equally critical for software project to have a strong user base in order to survive. The software teams have approximated the value they provide to users by trying to count users and downloads, but it is difficult to capture the varying value and role of heterogeneous users.

3.3 Power users help grow the project into infrastructure

An extreme case that demonstrates the differences among users is seen in the role of power users, or "friendly users." Power users have a close relationship with the software development team, and are known by name. They are expert and extensive users of the software, sometimes contributing code to the development effort. They are especially active in reporting bugs and giving feedback, introducing their requirements, and are often willing to test out early versions of the software before it is released. The development team becomes especially attentive to power users' voice in exchange for their help with feedback. These users are key champions of the product within the user community as well, teaching others about how to use it or developing special scripts and code that can help make use of the software within users' particular scientific field. These users are both experts in using the software and also have a relationship with the software team. Being a power user, as I use the term, involves a relationship with the team. Therefore the number of power users is limited by constraints on the software team's capacity for deep relationships with users. Software team members that I spoke with could typically name two or three power users immediately. Taking into account overlaps among those named by team members, the number of power users may be approximately equal to the number of members of the software team. This approximation must be taken with a caveat for the difficulty of defining a power user among the gradations of expertise and relationships that team members have with users. Indeed, counting power users may be even more difficult than counting users.

Definitions of power users aside, let us turn to consider the types of work done by power users. An ESMF team member describes the role of power users as follows:

> We have people who have contributed code. Who haven't been paid for it really but they tend to be on our, I mean they are still in our network of collaboratives. It's not like random people who contribute to code. But we've had lots of contributions from one guy within DOD who is implementing ESMF in bunch of DOD applications. He goes in and he's been working with our NEC file system recently and that's been great. We have NASA likewise, they have created capabilities that we've brought in as well. So our whole config, we have sort of a model configuration utility that came from NASA. We didn't have to write that. And some of the specialized functions like observations data streams, that came from NASA too. (Grace #2)

For ESMF, power users have developed parts of code that may be either cleaned up and included in the main ESMF distribution, or simply distributed "as is" by the ESMF team as something other users may find helpful. As "Grace" noted, this is not open source development in the mythical sense where hundreds or thousands of people are studying the code and submitting incremental improvements. Instead, access to the code allows specific groups to work with it and make local customizations and improvements for their own purposes, which may then be integrated into the larger code base.

The software team's relationship with power users often entails a personal relationship, too. In some cases, this is from a prior working relationship where a developer worked on a project with the power user. Other times, the relationship develops over time as the developer and the user communicate about the software project. Developers will sometimes do site visits to their most important users to hear about their needs and to help them work out issues they are facing. Users and developers meet face-to-face at conferences and workshops, allowing informal interactions over dinner as well. An NCL team member summarizes her interactions with one power user:

> Of course, there are people who come here for visits. For example, "Alex" of Purdue University, and he asked for things. He does paleo- types of things and he has all the students use NCL, and… to my knowledge, anyway. But, you know, if somebody like "Alex" asked for something, then it kind of comes right up to the top too, because of just personal relationships. (Marina #2)

Personal relationships are just one way that power users establish their position, and requests to developers are just one of the effects of their influence. In the following sections we will look in more detail at how power users come to have that role and what they do with it.

The dynamic of power users' relation to the software team is fundamentally one of reciprocation, a characteristic that De Roure and Goble list in their discussion of the relationship between scientists and teams developing scientific software (2009). Power users provide feedback, testing, code, and advocacy in exchange for improved support, a voice into development priorities, and good repute within the community. In the process, they become personally invested in the software's success and build relationships with individuals on the software team. Because of the reciprocal relationship, we can summarize this as "power users earn the attention of the software team" and "software teams earn the attention of power users," to illustrate the mutual exchange that occurs. Figure 5 shows this interaction.

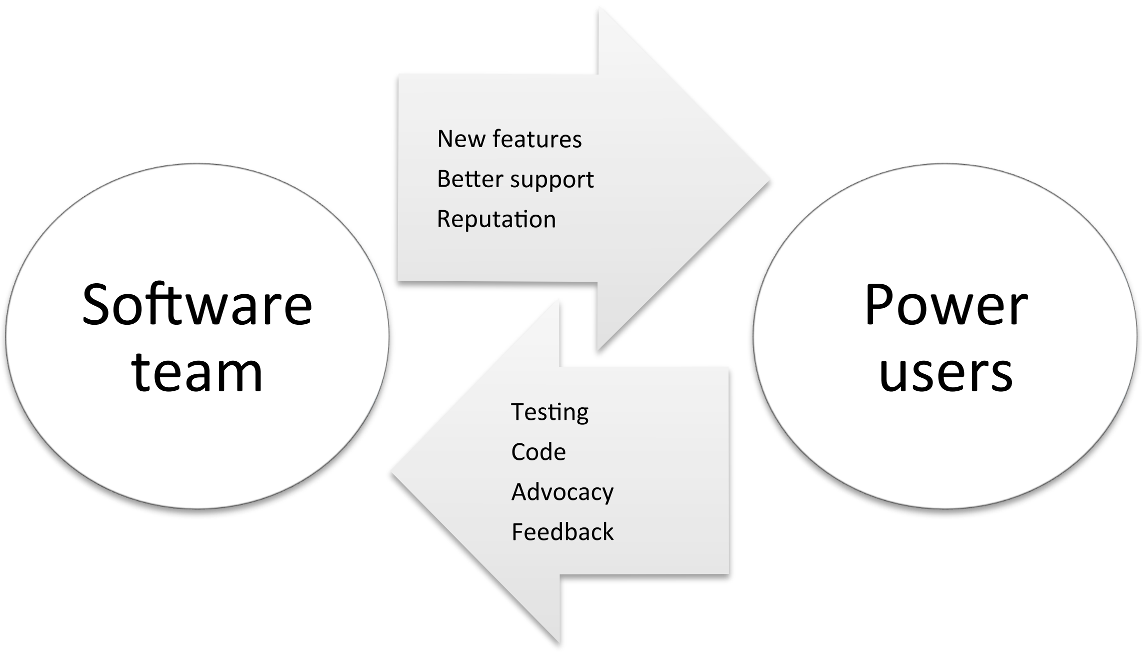


Figure 5: The software team and power users exchange favors.

3.3.1 Power users provide favors to the software team

Let us first consider what power users do for the software team, and begin with testing. Power users have a significant stake in the software since most have used it for a long time and/or it is a vital part of their work. As a result of their extensive usage, they have more opportunity to uncover defects in the software and are motivated to get the bugs fixed by reporting them. Their familiarity also allows very effective bug reporting, with a power user even offering to ESMF developers "As usual I can offer to write a reproducer" when reporting a bug (esmf_support email). In effect, they become important as testers. Their testing may not achieve the coverage achieved by the usage of dozens, or thousands, of less frequent users who report their problems. However, the relatively high amount that power users use the software makes them natural candidates for the software team to ask testing favors without bothering the larger user population. So when preparing a new release of the software, NCL developers give a few power users an early copy of the code, and ask them to do some further testing. The developers are concerned that "I have a limited data set to test…if we are gonna use other data set it may not work" (Melissa #1). The NCL software team wants to ensure that the software works for its users in their various situations. Power users are an accessible set of folks who can test real-world scenarios with more types of data than developers have, and make informed reports of their findings to the software team.

Power users also work to develop software that utilizes and extends the software project of interest. NCL power users write scripts for many different analysis tasks, and may share the most useful ones with others. Sometimes this happens within specialized subsets of the user community, like WRF, where power users have set up a webpage for sharing the NCL scripts that are most useful for working with WRF data (http://www.mmm.ucar.edu/wrf/OnLineTutorial/Graphics/NCL/). As this webpage notes, NCL has subsequently incorporated some of the most commonly needed WRF processing functions, so that they are automatically distributed to all users when they download NCL. The NCL team also hosts its own collection of sample scripts for commonly used tasks. While NCL developers wrote many of these, others are contributed by (power) users who thought that a script that they wrote might be useful for others. One CGD scientist spoke to the value of these contributions:

> But you know basically, it's an interaction where I can go downstairs and talk to [NCL developer] and say, "You know look, I've got this field. I want to do this." It's just so easy to do that, whereas, I've tried using Octave and even MATLAB to do some of this and it seems like you are reinventing the wheel every time to do this. Generally somebody working in NCL or somebody using NCL will have developed a script to do what you want. (Cameron #1)

Another example, mentioned earlier, is the configuration file for Emacs to display NCL code in a clearly formatted way. A power user developed this code to help create a work environment conducive to NCL use. In a similar way, ESMF receives code from a few users. This is sometimes code for a bug fix, which developers can import or reference while fixing the main ESMF code, or it may be code for a new feature that ESMF can offer for other users as an optional download. GSFC, for example, also has a whole software layer called the Modeling, Analysis and Prediction program Layer (MAPL) that builds upon ESMF and bills itself as “the fastest route to ESMF adoption” (http://maplcode.org). These layers of supporting software build up an environment that makes the software system much more robust and useful, and makes it fit practically with the needs of many users.

In addition to power users building an array of technical support structures for the software, they also spread the word to potential users and train novices. For ESMF, there may be a couple of power users among the application developers for each model. These individuals then educate others in their organization about the benefits of using ESMF and teach about how to use it. Even in the case of CCSM/CESM, where there has been some resistance to adopting ESMF, the model developers who work with ESMF can articulate to their peers the specific benefits they receive from it: time keeping, easier interfacing with other ESMF-compliant components, and possibly in the future, re-gridding. NCL has had even greater success with user advocacy, sometimes stimulated by the ncl-talk email list. On the list, users will answer questions for each other, and scientists who are thinking about using NCL can see the range and sophistication of NCL use. At NCAR, scientists report teaching each other how to use NCL, going to the known experts with their questions. Advanced NCL users become the “go to” data analysis person on group projects, and then their code is shared with their colleagues. At universities, this plays out among faculty and students. For instance, an advanced NCL user at the University of Colorado is known to encourage his students to use NCL and to show them how it can do what they need for their assignments. The advocacy and training that power users do are important because they signal that the community has taken the software as its own, and that it has become a significant part of practice in the community and is often learned as new scientists become part of the climate science community.

Both ESMF and NCL have asked power users to provide detailed feedback about the development of new features. The NCL team invites a few (power user) scientists to their monthly meetings, including someone from each of its two largest user bases: WRF and CESM. WRF is a larger community than CESM and now has more NCL users than CESM does, even though CCSM scientists began using NCL first. One of the NCL developers is also a scientist within CESM, so he ensures CESM’s needs are represented in the development plan. A collaborator from the WRF community is also invited:

> She doesn't come to our meetings so much. I usually meet with her individually because our meetings get too much, too detailed probably for her. I've invited her, but she's in another building, so I'll meet with her separately. She's kind of a whiz NCL programmer so she doesn't really

> need my help that much. [laughter] But occasionally, if they need some WRF function built into NCL then I work with her and do that. (Avi #1)

NCL's power users serve as liaisons to the scientific community, relaying current or upcoming needs. NCL's historically strong ties to CESM facilitate similar communication, so that the scientist who also works on the NCL software team is "just on this all the time because he's got his fingers on the pulse of that community and he'll say 'Look, "Avi," HDF-EOS5 is coming up on the horizon because we got a bunch of satellites going up and they're all producing HDF-EOS5 data coming out next summer. You guys need to be ready for that" (Avi #1). While the NCL-talk email list is valuable for expressing the current needs of the community, this sort of prognostic perspective on what new features NCL needs to support is very valuable to the team. In this way, the power users become the voice of the larger user base, articulating concerns or needs from scientists. This ability of the power users to represent a larger segment of users is extremely powerful, and NCL and ESMF are ready to listen to power users both to meet power users needs but also because they believe doing so will help them to meet the needs of many users.

3.3.2 The software team provides favors to power users

In exchange for assistance from power users, the software team grants certain privileges to power users. Engaging power users in feedback is a great example of an exchange of value for both parties. As power users present their perspective on features that the software team should pursue, those power users also have the opportunity to assert their preferences. An NCL team member describes this benefit for power users:

> …if they have a request, then we will put it higher... Higher priority than others because we give them... we weigh them as more important than others because they are giving feedback to us. We know that they are giving feedback to us. So for us, say if they have some request, so certainly we want to listen to their request. Well, he is like a friend so "if you help me, I'll help you." (Melissa #1)

This reciprocity makes the developers predisposed to listen to and prioritize feedback from helpful users. Such feedback from users comes more frequently than the timescale of larger requirements gathering processes, and also does important work in clarifying ambiguities in requirements or updating the requirements with the current needs of users. For this reason, both ESMF and NCL have solicited feedback from their power users at regular intervals and at particularly important moments. When ESMF developers recently proposed some changes to the way that they handle optional arguments in the ESMF interface, they sent a proposal out to the ESMF Joint Specification Team email list, which includes many users. But it is the power users who responded, raising concerns and even helping to propose a solution that would work better. The team heeded their advice, and went with a proposal closely related to one of the ideas proposed by the power users, fulfilling the "joint specification" part of the mailing list's name. In this way, ESMF's larger goal of guaranteeing backward compatibility in 2010 (as it had appeared in project plans) is made much more concrete with the help of a few users.

Power users receive much closer attention in support requests. Given the sophistication of the user, it is less likely that a question is trivial or due to a small misunderstanding. Power users understand what sort of problems should be elevated to the development team, and their support requests are likely to entail a software bug or feature request. Because of power users' personal relationship with developers, at least one team member will recognize their name when they submit support requests and will prioritize a response to them. In ESMF and NCL, power users directly contact developers sometimes rather than going through the official support lists. Power users also interact with the team through richer modes of communication. While most support requests are by email, power users sometimes meet with the software team by phone or face-to-face. ESMF will set up calls to talk with power users about specific issues, and NCL meets regularly face-to-face with the power users who are at NCAR.

For many power users, their reputation within the scientific community affects their career, collaborations, and funding. Positive publicity can bolster their perception by others. When ESMF refers to model codes that have done an exemplary job adopting ESMF, they can promote the expertise of the power users who were behind the effort. When NCL lists sample codes contributed by users, it helps build the reputation of those scientists within the community. So to the extent that the software team has a voice within the larger community of scientists and infrastructure developers, it can give positive publicity to power users and their projects. Reputation is less important for the software teams than it may be for academically-oriented power users, but software team members are happy to help bolster the reputation of power users as a thank you in exchange for the various ways that power users have helped the software project.

3.3.3 ESG lacks power users

By nature, some software projects will tend not to have power users. ESG is one of these, and so the above discussion of power users has drawn mainly from data about NCL and ESMF. There are a few reasons why power users have not emerged in the same way for ESG. First, the pattern in which users access the system is less frequent and less protracted. Users search for data and queue it for download, but once it is retrieved than they move into a new phase in which they work with the data without using ESG. Some of the most advanced data users skip the step of downloading data from ESG, since they often have direct affiliations or connections to the institution that is holding the data and can access it more efficiently elsewhere. These scientists also have less need for the search functions and metadata in ESG, since they know what they are looking for. For them, it has been helpful to be able to refer others to ESG when they get requests for data, but the scientists at the institution where the data are generated readily admit that "Quite honestly, I don't use the ESG myself" (Cameron #1). When one does not use the software regularly, it's difficult to become highly invested in it or develop other characteristics of a power user.

Like ESMF and NCL, ESG has focused its development around key classes of users. However, it has not engaged these users as part of the development cycle in the way we have seen power users function in ESMF and NCL. Instead, the ESG team has drawn up use cases describing the ways that it anticipates ESG will be used and what the system should provide in those situations. Use cases are helpful in generating new requirements for the features that should be implemented, and the use cases are informed by the concrete needs that ESG has for the scientists that is serving. As an ESG document summarizes:

> As a representative sample of these scientific needs, we will focus on requirements generated by the next IPCC assessment, the CCSM and its related SciDAC activities, and the Climate Science Computational End Station. These three projects are well aligned with the rest of the climate community and will provide a focus for ESG efforts and future development. (2006 ESG-CET Project Management Plan)

So the needs of these three projects are used as a proxy for the needs of the community more generally, and generating use cases for them helps to elicit requirements for the software. This is similar to the way that power users serve to represent large segments of the community in ESMF and NCL. However, ESG's approach to getting "feedback" or input from scientists makes it more difficult to iterate into a deeper understanding of the requirements and does not provide the other benefits provided by power users: testing, code, and advocacy. These other needs are at least partially addressed through data managers and scientists who are colleagues on the software team. As an ESG team member explains:

> …in ESG, [we] directly fund the Data Manager in the Climate Division. He's more classified as the software engineer, but he's embedded with the scientific group that does the climate change work. So, he manages our scientific data as opposed to... You don't want software people doing that. But he also is a liaison with the community, so he has a much better idea of how they work and what they want. (John #2)

The data manager approves all new users and handles the ESG support list, and so helps deliver feedback from and about the users to the ESG developers. He also uses ESG substantially in the process of doing this support, and therefore helps with testing. He's not a neutral voice in the community, so he doesn't have the same type of advocacy role that power users can. However, ESG does not need that in the same way as the other projects due to its monopoly in distributing certain types of data, ensuring that users will come to it. Furthermore, ESG is relatively easy to use compared to ESMF and NCL and does not require users to install software, and so it has less need for power users to help support other users and create a software ecosystem that integrates ESG into other systems. So although ESG does not have power users or the benefits that come from them, it also has less need for them and makes up for the roles that power users usually play through use cases and data managers who know the user community.

3.4 Conclusion: Power users are important but user counts are the goal

It might seem that the priority placed on interaction with power users is at odds with the role that large counts of users play in supporting funding and momentum of a tool growing in popularity. Might not the many “weak” users be more important to cater to than power users? Does attention to the power users’ desires risk sacrificing ease of use, since they have so much expertise? Do power users favor the software system so much that they become blind to its weaknesses? Fair questions, but there are several reasons that engaging power users takes priority over trying to engage masses of lighter users. First is a practical matter: it’s just much easier to speak with a handful of helpful users. NCL and ESMF both do try to get input from their larger user base, and have sent out surveys every three or five years to do so. However, that information is less frequent and up-to-date and not as rich an interaction as the team has with power users. Second, power users are seen to represent a certain segment of the user. So it is at least perceived that addressing the power users’ requests is tantamount to also addressing the unvoiced requests of a many more users. Third, there are times when the sophisticated applications of the software by power users are directly important for funding and popularity of the software. Shortly after the gulf oil spill in 2010, a few users approached the NCL team about getting some help making an animated visualization of the projected location of the oil spill. This single application of NCL can demonstrate its utility to sponsors, and help justify the projects value. So power users can have a disproportionately high impact on sponsors. Lastly, small-time users are not sufficiently invested in the software to develop a relationship with the developers and engage in the mutual benefits from doing favors for each other. It takes time to be gain expertise in the software and be helpful to the software team. While some users may have great expertise in using the software, they do not become power users (in the sense of it that I am using here) until they are helpful to the software team and engage in the reciprocal exchange that leads to benefits to both sides. Although software teams still value and seek to increase the raw number of users that they have, in practice they end up working with a few power users in order to do so. The NCL and ESMF software teams have found that working with power users is the best way to achieve increased adoption and to expedite their development work.

We have seen the importance and difficulty of counting users, and how user counts function as a metric for project success in lieu of revenue. There are many other possible metrics for project success (Olson et al. 2008b), yet user adoption rates are usually one of the first things that team members cite when communicating how their project is doing. Although determining the number of users is relatively easy compared to trying to measure some of the other outcomes Olson et al identify as indicating success, it is still not a simple or standardized measure. There is also a lag between when development work is done and when user counts reflect a strong software system. This is one reason why good project managers are so important for software projects: to stabilize the project while it grows and to free the software team from turnover that would disrupt the development of skilled and committed team members.

Despite the importance of numbers of users, I have also noted the disproportionate influence of a few power users. Power users develop a mutually beneficial relationship with members of the software team, providing feedback to the software team while gaining influence in setting priorities for new features or bugs to fix. This close relationship does not just benefit a select few. By listening closely to the concerns of a few users that represent larger sets of people with related concerns, then the software team can learn the detailed information they need to make a software system that will be attractive to large numbers of users. Most users do not have the time or interest to engage in detailed interactions with the software team. Power users can also train others and promote the software system in the daily practice of their colleagues. Along with being a practical way for software teams to engage users, interacting with power users functions to increase overall levels of user adoption. While counting users is an important and simple metric for project success, some users are far more important than others. Those interested in the success of projects should not get too distracted by raw user counts, but look to factors such as the quality of software teams' interactions with power users.

Chapter 4

Infrastructure Projects Prioritize Support

This chapter addresses how the project requirements – or goals and objectives – of a project are accomplished. First, I discuss how project objectives are articulated and describe the objectives of the scientific software infrastructure projects that I have studied. Objectives are most clearly stated in proposals or reports, so project managers, PIs, and sponsors are most conscious of them. These objectives are oriented around what the software should do. Users and developers are generally not aware of explicit project objectives, or at least do not think about them in those terms. Instead, they focus on what the members of the software team should do, particularly expressing that the software team should support the work practices and problems faced by users.

The second part of the chapter details how software teams go about providing support, and what that work accomplishes. Developers in the three projects I studied report that about half of their time is spent on support, which serves to both improve the software development process and to improve the process users go through in adopting and utilizing the software. The software development process is improved by support because it provides defect reports and updates the software requirements as developers have a ready channel for communication with users. At the same time, support helps to teach users about how to use the software and apply it for their particular needs. In short, it helps to provide a level of robustness and integration typical of infrastructure.

4.1 Official project objectives are put into practice via support

The most prominent official statements about each project focus on what the software can do for the user. In a sense, the types of software requirements that it satisfies. In this section we will see that the official project objectives are put into practice by focusing on the behaviors of the software team members, and especially their interactions with users. It is the process requirements then that come to mind most readily when team members articulate their objective.

In interviews, project managers articulated project objectives similar to the official statements on documents and webpages. This is as we might expect, since they author many those documents and are responsible for thinking about the project at a higher level. However, when developers were asked about project objectives, they often reported that the objectives weren't clear to them, or that they were not involved in that part of the project. Team delegate certain types of work certain people, and project managers think about objectives so that developers don't need to worry or think about them. When developers did articulate their understanding of the project's objectives, they had a different – although not incompatible – perspective than the project managers. Developers thought about project objectives as the motivation behind their daily work: serving the community of scientists. Below I compare the official accounts of project objectives with those articulated by developers. The three case studies are broken out separately to avoid confusing the different objectives of each.

4.1.1 ESMF focuses on helping certain users

> The Earth System Modeling Framework (ESMF) collaboration is building high-performance, flexible software infrastructure to *increase ease of use, performance portability, interoperability, and reuse* in climate, numerical weather prediction, data assimilation, and other Earth science applications. (Earth System Modeling Framework 2011, emphasis added)

ESMF's highest-level objectives have remained the same since the early days of the project: to promote interoperability between model components and to make it easier to build coupled models. At the very beginning, the software team undertook a large requirements gathering phase in which team members spoke with many scientists through teleconference calls and by email. The initial stage of the project thus revolved around software development and implementing a chunk of those software requirements, before it started "evolving" in response to new users:

> So the first five years is more of a development phase and then once it was developed and released, you start to worry about how to evolve and that's when a lot of the issues will show up. Of course in the development phase, the issues are more technical. When you get to the evolution, it becomes more of everybody has their own requirement and how do we converge those requirements. So it's more of a, it requires a lot more discussion and process. (Andre #1)

ESMF project documents recount the effect of this evolution as new models started using ESMF:

> These modeling systems introduced additional requirements, both technically and organizationally. The grid software delivered with the initial ESMF prototype had to be completely redesigned. The composition of the core team changed substantially, in order to bring in staff with greater expertise in mathematics, numerical methods, and application areas. The management structure of the project was reworked and formalized in order to accommodate multi-agency sponsorship. (Earth System Modeling Framework 2008)

The Change Review Board was one of ESMF's responses to the new organizational requirements. ESMF tends to be adopted through organization-level decisions when a modeling group decides that it will use ESMF, and in several important cases these groups have simultaneously committed to help sponsor ESMF. When a group makes this decision, the model developers leading the adoption effort soon become expert in ESMF – both power users and users with money. These roles tend to give them a lot of influence on the project, and when ESMF had accumulated several such parties it decided to make this influence more explicit and negotiable by forming the CRB. The CRB is technically open to anyone appointed by the ESMF Executive Committee, but "A typical CRB member controls the scientific and technical development of a set of ESMF-compliant applications" (Earth System Modeling Framework 2010). The CRB member can then individually express her preferences for how the ESMF software team allots its efforts and the CRB can help prioritize among competing requirements. The CRB is granted the authority for allocating the efforts of the ESMF software team. While this still tends to serve the project's big objectives of increasing interoperability and making model coupling easier, the more immediate objective is in serving the needs of users.

The ESMF team seeks to serve specific people rather than unknown users. Team members assume that the chances of adoption from unknown modelers who contact the team are small, as are the chances of getting help in return from them. So as one ESMF team member reported:

> we…targeted all of our development and all of our support to the [national labs]. Other framework projects, when they seemed to have not as defined a customer base that is one of the things that did them in, because they weren't targeting specific customers. They were just putting a product out, hoping people would pick it up. So, *if people supported us, we just knocked ourselves out to help them*, and we didn't spend a lot of resources trying to make it easy for graduate students. We just focused on the big labs. (Grace #4)

This prioritization is seen in the way ESMF developers relate to their users, from reviewing email drafts to have the best possible communication with key users, to designating specific developers as liaisons to develop relationships with particular users, to flying to meet with these key users and including them in the ESMF governance structure. ESMF developers report that they do not usually discuss objectives or goals. Nevertheless, one developer was willing to offer his perspective: "The overall goal is to make coupling models easier, to make building models easier" (Juan #2). To make modeling easier, the software team must both have a useful software product and provide effective user support. Powerful software that users cannot put into practice is useless. Another ESMF team member stated the objective as obtaining national or worldwide recognition (Edward #2). Gaining widespread recognition could be accomplished through widespread use, or by powerful and prominent use by a few. Recognition will critically rely on having usable products and providing users with helpful support interactions with the software team. *ESMF prioritizes some users over others, and recognizes that they are seeking to not just provide a software product but to assist these users with the process of modeling work. To do so, they need to not just provide a useful software artifact, but also need to provide support that helps the users deploy the artifact to improve their work.*

4.1.2 ESG serves other projects, and the users of those projects

> The mission of the DOE SciDAC-2 Earth System Grid Center for Enabling Technologies (ESG-CET) project is to *provide climate researchers worldwide with access to: data, information, models, analysis tools, and computational resources* required to make sense of enormous climate simulation datasets. (Williams 2009, emphasis added)

In its 2008 management plan, ESG specifies the "high-level requirements" that are driven by the "scientific requirements" of specific projects. Evolving from its initial focus on technology development, ESG now revolves around supporting the data distribution needs of a few projects: the IPCC led at PCMDI, CCSM at NCAR, and the Climate Science Computational End Station (CES) at ORNL. *When ESG becomes the host for data from one of these projects, it then gains a monopoly on data distribution for some set of users.* The scale of these projects drives the scale of ESG. So as the IPCC grows in reach and participation, ESG must "collect and distribute data on a much larger scale" and "extend metadata schema" to cover data from new types of model components. These steps still involve technology development, but are more targeted toward providing services than ESG's early days when the team just wanted to see if they could move large amounts of data over the network between national laboratories in the United States. These objectives do not indicate much concern with adoption or acceptance by the user community; they feel like that is guaranteed on account of ESG's designation as the data service for these major scientific projects. However, does think about how to support the (assumed) adoption of a community by making the service scalable, efficient, and usable. ESG currently sees itself as responsible for delivering data and helping users to correctly understand the data. So while originally ESG's main objective was technology development, its focus is now on supporting climate science, and particularly in supporting the data distribution needs of specific partner projects. The software team, which includes data managers and others that are involved in the partner projects, generates extended use cases for these particular projects. These use cases drive development, both meeting the immediate needs of ESG's partners and serving as proxies for the needs of the extended community of potential users.

ESG's emphasis on serving the needs of partner projects (including the expansive needs of CMIP as a "project") calls attention to two different types of ESG stakeholders. ESG works closely with the modeling groups that supply its data, such as by mailing 1 terabyte hard drives back and forth during the "upload" of CMIP3 data to ESG. By taking steps to support the "data providing" users, ESG secures its position as the designated mode for distributing data to its "data consuming" users. Developers on the ESG software team, though, do not interact much with the users who access data through ESG. Instead, the developers rely on the data managers who are members of the ESG team. These data managers interact with scientist end users by handling most of the support requests, and relay what they know about users to the software team. Users interact with data managers, and data managers interact with the rest of the software team. So users and the developers primarily interact in a secondhand way. One ESG leader emphasized to me the importance of these data managers, and is sure to pay part of their appointment out of ESG funds to ensure their time, attention, and feedback to the software team. While ESG's main objective had been technology development, it has recently focused on supporting the major projects such as CMIP that rely upon it. Practically speaking, it is just trying to meet the needs of the people running these projects, and relies on data managers' support and feedback for the "captive" user base.

4.1.3 NCL aims to provide exceptional support

> The NCAR Command Language (NCL), a product of the Computational & Information Systems Laboratory at the National Center for Atmospheric Research (NCAR) and sponsored by the National Science Foundation, is a free interpreted language designed specifically *for scientific data processing and visualization.* (NCAR Command Language 2011, emphasis added)

NCL has a much more loosely defined project objective than the other two, perhaps in part because of its position on core NCAR funding rather than having to articulate objectives to a review board while submitting proposals or reports. NCL is intended to address a certain stage of scientific work: post-processing. Rather than a statement articulating an objective, NCL's overview states the functionality of the software. NCL team members explain the three ways that the software provides this: file Input/Output, computational methods, and visualization. In practice, NCL's operations are based on certain other objectives beyond just producing that software. The gentlemen's agreement with CGD more than a decade ago amounts to NCL and NCAR's scientific computing division guaranteeing to meet the post-processing needs of CGD scientists. So in one sense, that is their main objective, and secondarily they seek to support the needs of all the other scientists who use NCL. More often than not, these objectives are not in conflict. Supporting the needs expressed by scientists range from doing small things like fixing a bug with coloring in a graph to larger tasks like converting to 64-bit addressing so that NCL can handle dataset variables larger than 2 gigabytes in size. The NCL team's emphasis on support though, indicates both the project's relative maturity and its focus on helping the users "upstairs" to do science.

Like ESMF, NCL emphasizes support. In fact, early on NCL team members determined that stellar support was one way that they could distinguish themselves from their competitors. As one team member reports of their conversations soon after the post-processing tool competition at NCAR:

> I said, "One of the things that NCL could do that would really separate it from the pack is to have really good responses to questions that come in." And there was kind of a loose thing that within 48-working day hours, a question would be answered, by somebody. And that to me was one of things… it's something that we hear all the time: the support is great. (Marina #2)

The team seeks to address all incoming support requests, but also recognizes that inquiries from people within the CESM community or known friendly users get priority. And within those, their colleagues upstairs in CGD get priority on account of the historic relationship with that division. This emphasis was especially strong when the project was young, and has weakened some as NCL has grown to include many users from many different groups.

This emphasis on user satisfaction as the objective is evident in comments by others on the NCL software team as well. One developer noted that he is uncertain of specific project objectives, "But at least NCL is to serve the field of science" (Melissa #1). Another emphasized the importance of usability, seeing the objective as: "[creating] a free, open source visualization package that [is] easy to use" (Patrick #3). The developer who gave one of the more official accounts emphasized the need to both help users with support but also to figure out what they need so that the team can create useful software:

> Well, our main objective is to support our user community, of course the NCAR user community and thekind of science that we do here, and then we kind of branch it out to the US universities and then the research sites. And so our first goal is certainly our US universities and the local users, so our main objective is to provide post-processing support to that group. Our goals are of course to be able to keep doing that. And they're not officially stated anywhere but I think a lot of our main goals is to try to stay on top of that community and be aware of their requirements. (Avi #1)

Yet even after this account of the project objectives that is already very user-centered, the developer felt compelled to add, “I think my own personal thing is just to provide that support and to never forget about the user and to always remember them when you're developing software. (Avi #1)” NCL’s broadest stated objectives involve providing a certain type of climate science software, but in practice they do this by making it their objective to prioritize their support of users’ work.

4.1.4 Users do need worthwhile software

The main objectives that software team members speak about – adoption and support – are also quite related. The NCL team has repeatedly emphasized this relationship, striving for excellent and committed support in order to spur adoption. Yet great support only gets the software project so far, it must also have a great software system. Serving the community entails making software that helps them do scientific work better or faster. ESMF has emphasized the use of ESMF infrastructure in order to save redundant work and make model development more efficient. For example, with ESMF, there’s no need for every modeling group to implement a regridding scheme since ESMF has already implemented regridding algorithms. Then ESMF superstructure creates technical interoperability between model components, facilitating coupling and the generation of new types of models. Similarly, NCL seeks to make scientific work easier by providing tested implementations of common analysis and data manipulation routines, along with a library of NCL scripts that users can adapt. ESG enables access to model data to a larger and more diverse set of scientists than was previously possible by cutting out many of the obstacles to getting permissions and finding the right data on supercomputers. These are each examples of ways that the software team seeks to make a product that is very practically useful and thus make adoption worthwhile.

Due to this need to both have ample software support and also worthwhile software, the software teams that I have studied split time between both types of work. Both aspects are important for adoption and increasing user counts – that important metric used both as a proxy for scientific impact and for determining funding merit.

4.1.5 Some stakeholders want to produce research instead

Among the three software infrastructure projects, there was a notable exception to the predominant narrative of creating useful software for the users. That is, some stakeholders do not need worthwhile software. More precisely, for these stakeholders worthwhile software involves coding work that advances computer science research. Early on, ESG focused on technology development and computer science research around the idea of using grids to move large amounts of data around high-speed networks. When ESG began distributing climate data, one of the team members arranged a usability study to go out and get the perspective of the intended users: climate scientists. This perspective brought several revelations, including an awareness that scientists tended to prefer browsing through data sets to using a search interface. Since then, the software has worked to incorporate a faceted search that allows aspects of both search and browsing. However, it was only when the project's emphasis shifted away from pure technology development to building useful infrastructure for climate scientists that they sought out feedback from their users.

The branch of ESMF working with the Earth System Curator project had an issue with differing project objectives as well. The award was distributed across several different institutions, each with their own pool of money and personnel. The ESG portion of the project was more interested in producing software of production quality that would be ready for the large number of users ESG anticipated as part of the CMIP efforts. Other institutions within ESC, however, were more interested in research results. Team members from these institutions were glad to prototype an interesting contribution for ESMF, but were not interested in spending the significant time to clean up the code, integrate it into ESMF, and support it for users. So while infrastructure projects tend to be more about supporting and helping users as chief objectives, there can be certain phases or segments of the project that have different (and conflicting) objectives.

The infrastructure projects at the core of my three case studies have largely avoided the computer science research efforts that often produce tensions in scientific software projects. Instead, they have stayed focused on their goals of providing software that does certain things, and in making the work of scientists easier in some specific respect. We have seen that the software teams aim to put this into practice by providing good support to their users. The latter half of this chapter details what software support accomplishes, focusing particularly on the way it facilitates adoption and helps developers to understand software requirements.

4.2 Support leads to adoption and understood requirements

My data show that the non-programming activities of the software team are key for both adoption of scientific software infrastructure and for the requirements engineering work on this software. Of course the development work that goes into producing software is essential, but a variety of "ancillary" activities are actually essential if the software is to be used and suitably constructed for use. This section outlines the different types of work done by the software team, and then focuses on describing the effects of support on adoption and requirements.

Developers that I spoke with spent about half of their time programming, and spent the rest of their time doing support, documentation, training, meetings, or other tasks. ESMF developers reported that about 50-60% of their time was spent doing software development, and the rest on activities like support. Although the exact proportion varies among members of the software team, some NCL developers attributed as little as 20% of their time to programming. This shift to even less development and more support-related activities reflects both NCL's emphasis on support and the relative maturity of the software project. It now has a larger base of users to support, and has already implemented the key functionality that it intends to include, with development focusing more on special initiatives or relatively small adjustments for users' needs. ESG developers spend relatively more time on development, since the project is in an earlier stage and a data manager handles much of the support work. Although it varies based on the project's stage and across the different developers, overall, developers report that non-programming work takes about half of their total time. The large role of non-programming work is consistent with literature about commercial software development, which also indicates about 50% of developer time being spent on programming (Perry et al. 1994; Vessey and Sravanapudi 1995; Perlow 1999; Goncalves et al. 2009).

Non-programming activities are not always distinct and cleanly separable from programming work. Documentation tasks may be intermingled with development. Support requests often blur between trying to help a user to understand the software, and checking or fixing the software code to address a bug. As an ESMF developer recounts:

> When [user] was putting in the re-gridding he found several bugs in a row. I mean, when he was using the re-gridding. Well actually, I think there

> was only one bug but he had a lot of trouble sort of figuring out how to use it and so, there was a lot of back and forth about that and trying to get him tested to figure out what was going wrong and sort of recommend solutions and stuff like that, so. (Juan #2)

Juan needed to both help the user and also understand that ESMF is not perfect software either. So the support request bled into debugging ESMF code as well as checking the user's code, and ultimately required some development work to fix the problem in the ESMF code. NCL users recounted similar situations. So we find much blurring between working alone and communicating with others, and between support and development, which hints at the ways that the different types of work complement each other.

4.2.1 Support engages users and provides training

Software developers engage in a variety of non-programming activities. While team members such as project managers entirely do non-programming activities, here I focus on the work done by software developers. The line between development and "other" work is somewhat arbitrary, as all the work contributes to the development of the software system. However, developers perceive some work as more technical, or as the "real" work. The activities that I identify are roughly aligned with this perception, and are culled from informants' responses when I asked them about what else they do "beyond development."

ESMF acknowledges the significant role of these activities by planning only 20 hours/week of development work when scheduling tasks, assuming that the other half of developers' time will be consumed by other work. Such work includes both unplanned and pre-scheduled tasks. Table 3 lists the most common activities, broken down into categories. Support involves responding to user inquiries and interacting with users (be it by email, phone, or face-to-face), while documentation need not involve direct interaction with users. All three projects have websites that they operate, and additionally, ESG needs to maintain major operational systems. Production of papers and presentations involves both creating new scientific knowledge and publicizing and promoting the software. Finally, there are a number of administrative tasks developers must do to get paid and have access to the appropriate rooms and computers.

Support • Help – how do I use (primarily technical content) • Help – how do I apply (primarily scientific content) • Feature request • Bug report • Training and workshops for users
Documentation • Help pages • Examples • Tutorial materials
Operate the webpage or other systems
Produce papers/presentations for research and publicity
Administrative: e.g., safety training, timesheets, getting computer access

Table 3: Non-programming activities of scientific software developers.

Work that is not programming, but which informants still classified as "development" work, is not recorded in Table 3. This includes documentation of the code for other developers, supporting and coordinating with other developers on the team, testing the software, and working on software design either individually or in a meeting with others. The literature documents the large quantity of time that developers spend doing this coordination work (Curtis et al. 1988; Walz et al. 1993; Herbsleb et al. 2000; Goncalves et al. 2009). However, I am more interested in the split between the work related to producing code and work in producing other things, as opposed to the divide between work producing code individually and work producing code while coordinating with others. .

There are a few additional types of work developers do that I did not include in Table 3 because they are irregular and rare. One example is the usability study that an ESG developer designed and ran with some users. The need for this activity I interpret to indicate insufficient interaction between developers and users, perhaps because of ESG's support system, which directs emails first to a data manager. These irregular activities can still be quite important for a project. However, they did not come up much during the interviews, and so my focus on non-programming activities will be for ones that are more regular or frequent in nature.

For the remainder of the chapter we will focus on the vital role of support work, as this is the activity that involves interacting with users. Most of these interactions happen via email support lists, although support is occasionally provided by phone or face-to-face. Training workshops function as a concentrated and organized dose of support work, allowing both instruction of users and giving the opportunity for the software team to hear from users audibly and immediately about what is working well for them with the software and what makes users struggle with the software. By engaging users, support connects scientists to the software and helps teach them how they can be come successful users.

4.2.2 Support engages developers and helps refine requirements

Once the project moves into a phase where it has users, support is a very important mode of interaction to hear back from users about their needs and problems, and to get feedback that helps to elicit new and elaborate on existing software requirements. Some support requests directly lead to or propose a new software requirement. Here is a simple example of this sort of exchange on the [ncl-talk] email list between a user and a developer:

```
It might be possible but it would not be very
convenient or easy with the current version of
NCL. Image import is on our to-do list though.
-dave

On Feb 8, 2011, at 10:00 AM, ___ wrote:

> Dear All,
>
> Thanks for reading this E-mail!
>
> Is there any function in NCL to convert img
> file to NetCDF or HDF or Grib?
>
> Thanks,
>
> Xiaoming
```

(Email on 2/8/2011)

Users politely inquire about whether a desired feature is available, leaving it either for someone to clarify how they should go about a certain operation, or failing that, making it readily convertible into a request for an easy way to do a certain operation. In this case, the general category of problem was already under consideration by the developers and so this inquiry then serves to reinforce the need and make more specific the usage scenario. The software team's list of feature requests is not public to users, so they will receive multiple inquiries for some features. This simply serves to reinforce the need: "the request list for us becomes a to-do list. Then after a while we can see that if a lot of people ask us for this feature and it seems to be important, then we probably... We will try to implement it" (Melissa #1). In this way helping users by answering questions and updating requirements for the software are closely entwined. Furthermore, through the open support lists most users have access to the requirements gathering process.

Bug reports also serve to extend or clarify requirements that are not properly implemented. Here NCL already had a requirement for a function that would return the distance between two latitude/longitude pairs. However the bug report says that this function is not being satisfied in all cases. Or, alternatively framed: there is a new requirement that the function should also work on the Cray XE6 platform. The exchange between an NCL user and developer follows:

```
THX ... To my knowledge we do not have access to
an XE6. Still, we could add some code that checks
if [rlat1,rlon1] are exactly equal to
[rlat2,rlon2] and, if so return 0.0

Cheers

On 02/09/2011 06:15 PM, ___ wrote:
> Hi all,
>
> I think I may have found a platform-dependent
> bug with gc_latlon. I've found that when my
> input latitude and longitudes are the
> identical, rather than having a 0 returned (as
> in a distance of 0 km between the two points),
> occasionally I will get an NaN returned. I only
> see this issue if I'm running on a Cray XE6 -
> when I run the same code on my local linux box,
> I get a 0. In both cases I am running NCL
> 5.2.1.
>
> Here is some example code.  NCAR NCL folks - I
> have uploaded the two .nc files referenced in
> the code below to the CGD ftp site if you'd
> like to try to replicate what I am seeing.
>
> Matt
```
(Email on 2/10/2011, emphasis added)

Getting access to the great variety of computer platforms used by scientists is a recurring challenge for NCL and ESMF developers. (ESG mostly escapes this by virtue of being a web service.) In this case, the problem was simple enough to be fixed without figuring out how to get a developer an account on a Cray XE6. In other cases, it is not that simple. While initially the development priorities are dominated by a plan and vision of the software team, accompanied by a formal requirements document, over time the project is owned and influenced more directly by user requests and needs. An ESMF developer documents this transition over the course of the software project: “We have this task list of things we need to put in and I think a lot of that was developed by the early kind of users who kind of decided what the project should be and how to do one of our early design stuff they did, so I guess it's all kind of user needs. But now we have feature requests that are more coming in from users now” (Juan #2) This developer notes that all the requirements are rooted in user needs, yet there is a sense in which the early collection of requirements is a different activity because its intertwined with the project identity and fundamental design decisions. These initial requirements are accumulated into a substantial set before launching into the development work. Then as these are gradually completed, the project rolls over into a more responsive mode where the team evaluates new requests as potential requirements for the software.

Power users frequently utilize the support channels. Below is an example of a message from an ESMF power user. This email takes a very different form than that of the user who is wondering whether a feature exists, and if not, implicitly requesting it. Instead we see a deep understanding of the ESMF software, and an established relationship with a developer on the ESMF team. Some technical portions of the message are omitted for sake of brevity, but it is easy to see the level of understanding and depth of reporting of the bug that makes it easier for the developers to follow:

```
Hi [first name of ESMF developer],
```

It has been a while since you and I talked. Few days ago I came across some bugs/unintended behaviour that I would like to report.

One of the problems has to do with "recursion" - when a gridded component executes comp method within a comp method (this is still the same GC, not a child of the GC), and the other issue is an usage of conditionally initialized variable.

... Last week somebody repored (sic) segv when calling ESMF_ReadRestart, and this morning I took a closer look. This is what happens:

If a gridded component executes ESMF_ReadRestart from within its registered Initialize routine, the address of VM cargo that is part of the original ESMF_GridCompClass is changed to the adress (sic) of VM cargo of the ESMF_ReadRestart routine at line 647 in file ESMCI_FTable.C. ... ***Currently, the only way to work around this*** is to pass BlockingFlag=ESMF_NONBLOCKING in the call to ESMF_ReadRestart (which bypasses the call to c_ESMC_CompWait).

However, this workaround exposes another bug: the userRC is set to localUserRc ***at line 1032 of ESMF_Comp.F90...***

I believe the behaviour described above, flags a design issue...

At Goddard, we still run GEOS-5 with ESMF_4_0_0rp1. So all of the line numbers correspond to the tag ESMF_4_0_0rp1. I have not tested any newer ESMF versions yet!

Please, let me know of your thoughts and if any of this is already fixed.

-[first name of user]
(Email on 11/8/2010, emphasis added)

About one hour later, this power user received a detailed reply, explaining that the first part is indeed due to a bug, which will soon be fixed, and asking if his group will be able to upgrade to a newer version of ESMF in order to obtain the fix. We can see that there is a way that these power users are providing second-tier support for the software team. This user is a model developer, and he tried to determine the problem that one of the model users was having before inquiring with ESMF about a problem in the modeling framework. Power users provide a more tailored level of support for scientific users in their area, and also help translate problems or needs from "their" portion of the user base into the easily digestible form of a bug report or feature request to the software team.

Support requests lead not just to new software requirements, but can also lead to requirements for other aspects of the software project. ESMF users have asked for better documentation, pointing the software team's efforts to areas of the documentation they see as weak, confusing, or missing altogether. One NCL developer describes the reasoning process he used when determining whether to turn a support question into a documentation task, creating a detailed example for the NCL web page:

> And just yesterday, I spent probably 5 hours writing a script for a user that asked a question, but it was a complicated script that I knew that he wouldn't be able to write. [laughter] And it was one... I try to weigh and I'll say, "Well, is this a question that somebody else is going to answer?" "Is this a popular thing that we get a lot?" And then, if it is, in the case of yesterday, it was, it was a popular thing. I said, "Okay, I'll write this. I'm going to put it out on our examples page and make it available to everybody." So, it's not just, "Oh, one user got 5 hours of my time." (Avi #1)

This important documentation work helps to build out the collection of artifacts that make it practical to utilize the software in the day-to-day tasks that users encounter. Addressing such requests also builds out a more detailed understanding of the ways that users are trying to use the software. One ESG developer reported getting frustrated when interacting with users because of the difficulty in trying to understand what users mean. He attributed his frustration to both a lack of understanding about the scientific content and a desire to work on technical stuff rather than work with people. After a period of low productivity another team member stepped in as a liaison to the scientists with whom he worked. This team member helped to translate the scientific needs into software requirements for the developer. In cases where users interact with only one or a few team members, those few people then have a critical role in interpreting and routing user communication to the correct people within the software team. NCL and ESMF directly routed all support requests to the whole software team, but ESG used a liaison model where a key person – usually a data manager or another person with some degree of scientific knowledge – is responsible for handling support and answering it or relaying support work to developers when appropriate.

For support requests that involve bugs or feature requests, there are a number of significant steps in processing each request after the software team receives it. All three projects use ticketing systems to track bugs and feature requests. Web-based ticketing systems are intended to clarify distinct tickets, include a little metadata about the item (e.g., creation date, completed date, creator of ticket, assignee to complete the ticket), ensure that none are lost, and provide easy access for everyone on the software team. The ticketing system helps present all outstanding items to the software team so that the project manager or whole team can prioritize work. Smaller tasks tend to be prioritized at the discretion of the software team, and need not wait until a larger team meeting. Meetings to sort through requirements are held every 3-4 months in both ESMF (by the CRB) and NCL (by a manager and the software team). These meetings review recent development progress and outstanding development tasks to prioritize what should be done next. Through this process of sorting and prioritizing requirements to implement in the software, the team aggregates the desires of many users and chooses what should be incorporated into the software infrastructure. The software team has a larger perspective of the whole software system and the full range of users, so team members seek to define more generalized versions of user requests. As one application developer reported about ESMF, if the modelers were to develop the code that they need in a modeling framework, they would make it too specific to their particular needs. By generalizing the software so that it can work for different users, moving it closer to the status of infrastructure that a community can depend upon.

While interactions during support are one of the primary places for requirements elicitation, they are not the only place where new requirements are determined. Some requirements are determined by project managers based upon their interactions with scientists, or are written into new funding proposals. Community meetings, primarily used by ESMF, were targeted attempts to get together with a selection of users for feedback and updates on the software. Other times, developers themselves add software requirements as they are programming and find a bug, think of a capability that could be useful, or determine more detail for an ambiguous requirement that they are implementing. Support interactions about requirements are particularly valuable, though, because of the direct communication with users and the opportunity to clarify the requirement in whatever detail is needed. Support makes requirements work efficient by providing a mode for clarifying and updating the understanding of requirements, as recent the requirements engineering literature has repeatedly emphasized as important (Wiegers 2003; Damian et al. 2010) . The need for updating requirements through support parallels principles of effective communication that use continual exchanges to build up common ground and common understanding, rather than relying on "one-shot" interpretations of long communication messages (Clark and Brennan 1991). Support provides an avenue for just this sort of rapid exchange to build effective communication between the software team and users.

4.2.3 Support makes a user

The support that developers provide also plays an important role in facilitating adoption. The software teams' interaction with users during support provides an opportunity to learn who the users are. As we have seen, software teams engage with users not just to learn about them but to help coach them about how to use the software system properly or to apply it in their context. Woolgar observes a similar phenomenon in his study of usability tests by the manufacturer of a new computer, which emerge all the more strongly in the more extensive interactions that ESMF, ESG, and NCL have while providing support.

> This paper takes the line that the emergence of a new range of microcomputers crucially entails the definition, delineation, and emergence of The User. We could say that this process amounts to the (social) construction of the user. However, it is not just the identity of the user which is constructed. For along with negotiations over who the user might be, comes a set of design (and other) activities which attempt to define and delimit the user's possible actions. Consequently, it is better to say that by setting parameters for the user's actions, the evolving machine effectively attempts to *configure* the user. (Woolgar 1991)

His key insight here is that the "social construction of the user" is not just about deciding on the identity of users, but its also about constructing users by changing users' practice of science by introducing new possibilities and constraints. Constructing the identity of users is a matter of deciding who are your priorities for support: power users?

users from favored projects? users from the United States? As Ribes and Finholt frame it, this is an act of determining the constituency that the software will engage and accommodate (2008). And as we saw in the last section, *supporting certain users will further push the software to address the requirements of those users. The software will support their uses.*

The software teams have great ambitions for configuring the user by changing and improving the scientific practices of users. This is an essential piece of the cyberinfrastructure vision, as reflected in this comment by an ESMF team member describing her idea of project success:

> I guess the best case scenario simply would be that's [ESMF is as] pervasive as NetCDF is now, it's just sort of a low level community standard that people contribute to and accept and use. And it's just become kind of routine. That's on the infrastructure side. I guess on the application side it would be nice to be able to grow the project out so that as we see needs emerging, and climate impacts and all those areas of emergency response, that some of these capabilities that we've been exploring like the work flows and getting products out and getting weather products out in ways that are really well documented, turns out to be useful to people, helps people to make use of them that they can find information, they can understand and they can use it. Those are sort of the simpler goals. And then the more complex goal is trying to sort of work on transformation of the community socially. … So I guess the longer-term vision is trying to build up the global community that is really using global assets and is working collegially together. (Grace #2)

Over time, ESMF believes that the way to advance climate science is to configure scientists to have modeling practices that rely on tools like ESMF, and reap the benefits of this new community configuration. At a smaller scale, NCL is accomplishing the same thing: transforming user practice to rely on NCL as a tool. As an NCL user within CGD says:

> What I like most is the support from the NCL team, they are very responsive. I also like the fact that people in my group are using NCL and I can ask them if I have problems. (Josh #1)

NCL's support expands the user's ability to do science with it, allowing him to build a set of scripts and analysis methods that rely on NCL. (Chapter 5 will further discuss how support can directly impact scientific practice.) Along with the users' comfort and knowledge about NCL, this then constrains the users' future actions to favor

NCL. Furthermore, the CGD group has achieved a critical mass of colleagues using NCL, so users can ask each other questions and share useful NCL scripts to do the analyses they need. The NCL team's intensive support of CGD has not just resulted in adoption by individuals, but created a self-sustaining unit or culture of NCL use. Scientists who are new to CGD are by default trained into using NCL, even without any actions taken by the NCL team. So as the software team provides support to individual scientists, it configures them into effective users that learn how to use the software in conjunction with their particular needs and the other tools that they use. In turn, configuring individual users makes the community more favorable for using the software system, building an ecosystem of users and supporting scripts and software that all integrate with the software system.

4.2.4 Building infrastructure means providing support

To illustrate the importance of support and continual engagement when building working software infrastructure, consider the following episode in ESMF. In late 2009, an Earth System Curator researcher presented to the ESMF team about a prototype program that automatically generated couplers for ESMF. The demo went well, and impressed the developers. The team was excited by the new possibilities of the software, and talked for an hour or so about how users might go about using and what some good extensions to the software would be. In the course of this conversation, it was clear that the researcher did not have a good sense for what users might need, wondering things like: "It would require talking to programmers, scientists to determine what makes is needed for them." Those conversations with users clearly had not yet taken place. The end of the presentation turned more practical. How could the researcher transfer the software to the ESMF team, since he does not currently have security permissions for the ESMF systems? They decided that transferring a tarball with the source code would be fine for now, even if that does not allow the ESMF developers to run the software right away. Then came a series of additional questions: how is the code licensed? Is the researcher willing to assist with development? What about development that has deadlines? The researcher replied that he has no problem sharing the code and putting it into a state that's "usable by different institutions." However, he would not commit to meeting any deadlines. An ESMF team member replied, "That's fine. When you do prototyping things, that's great. But there are always these issues when you go to use it. Bottom line is, it looks really great." And implicitly, bottom line is that we are going to put it on the back burner.

The prototype code has not been incorporated into ESMF in the 18 months since that presentation, nor is it included in ESMF's tracking system of things to do. Yet even if it is never incorporated into ESMF that does not mean it was useless for either ESMF or the researcher. The ESMF team gets free publicity for its framework among all the audiences to which the researcher presents his work, and was able to include this research partnership in the 2009 ESMF annual report (Earth System Modeling Framework 2009). And it is possible that the code or ideas *may* still be useful within ESMF at some point in the future. The researcher only needs a prototype in order to publish his findings, and ESMF provides a nice real life case to motivate his research. Both parties can walk away happy, but neither would argue that the prototype coupler auto-generator has become part of the software infrastructure.

Accounts of prototypes developed by computer science partners have relatively rare in the infrastructure-oriented projects here, but they are common in the development of scientific software more generally. The issue can be explained by considering the very different project objectives: computer scientists' aim for the software is to produce publications that advance their research agenda, in contrast to the aims of adoption and helping users that we have seen to be the emphasis of software infrastructure projects. Both types of projects have important activities that complement the software development work that they do. However, *for infrastructure projects the complementary activities are dominated by support, and for research projects they are dominated by publishing.*

Infrastructure projects are not opposed to publishing or creating scholarly knowledge based upon their work, but it is not their focus, just as research projects are not opposed to doing support work, but will not give it much attention. High-quality support is rewarded and of interest to professional software developers that are involved in infrastructure-oriented projects, and high-quality publishing is rewarded and of interest to scholars working on research-oriented projects. To the extent that the software teams studied here have engaged in research work, the resulting code is generally not seen as infrastructure, but rather as a "pre-infrastructure" that could later be migrated into the software system depending on the priorities expressed by the users. The tension of research vs. production systems (Weedman 1998; Lawrence 2006; Ribes and Finholt 2007) in large part is a consequence of whether the team spends its non-development time doing support or publishing. As we have seen, support work is needed to refine requirements, put the software into practice, increase adoption, and help tailor the software along with a collection of trained users and cooperating software systems. So whereas support gives feedback about the requirements the software needs to meet the users needs, the process of publishing scholarly work yields requirements to meet the needs of producing novel research. Research dominated efforts thus lack the support work that is vital for turning a scientific software system into scientific infrastructure.

4.3 Conclusion: support is critical for software infrastructure

Woolgar's account of the computer engineering process depicts engineers building a system with one type of user in their minds, and then the engineers are confronted with the reality of their actual users during user testing (Woolgar 1991). The software teams I have studied here had similar experiences, of suddenly having to revise the software requirements in light of a better understanding of users. NCL offers a clear case of this phenomenon, when they had to rewrite the interface to NCL into the functional programming paradigm scientists were used to instead of using an object-oriented programming paradigm. User support is a critical activity, because it helps to gather information from and about users, calibrating the software teams understanding of users' needs. Support can help to avoid jolting the requirements around by providing an open channel of communication with users that can continually update requirements as users put the software into practice. Given the costliness of pursuing the wrong requirements or misunderstanding requirements (Boehm 1988; Wiegers 2003), support work is critical for keeping the software project cost effective. Support work helps to improve both the software team's and users' understandings, a process of "infrastructuring" (Pipek and Wulf 2009). As Pipek and Wulf describe in their model of infrastructuring (see Chapter 5 for a more detailed discussion), making infrastructure involves incremental tweaking of both user practice and the software product. Support provides the communication channel that allows incremental tweaking as users and software team members converge on an infrastructure. For scientific software that aspires to become infrastructure, active user support constitutes a vital team practice.

Ribes and Finholt document case studies where cyberinfrastructure design is dominated by science questions, leading to teams that prioritize research first, then software development, and lastly maintenance work (Ribes and Finholt 2007). The case studies that I have studied reverse these priorities, and have more infrastructural goals. User support is a key component of doing this, since it accomplishes maintenance work and contributes to software development. Support work helps update requirements so that the team builds the right software and helps users to make effective use of the software in their daily work. Sponsors have tended to look more at the research outcomes of software projects (in Ribes and Finholt's cases), or the characteristics of the software itself (in my case studies). Instead they might also look to the user support strategies of the software project, recognizing high-quality support as a predictor of a successful project. They could also work to development metrics based upon the software team's user support, enriching their picture of adoption beyond simple user counts.

Chapter 5

Making Stable Infrastructure Amidst Changing Science

Scientific software, or at least parts of it, has many of the properties of infrastructure.[1] This chapter analyzes the role that software plays within the scientific knowledge production system. Both my informants and the literature conceive of scientific software as an infrastructure that can be transparent and relied upon to facilitate and ease the production of scientific knowledge. Here I develop the relationship between infrastructure and scientific knowledge that was introduced in Chapter 1, propose a simple model for describing scientific software infrastructure, and contrast it to other models that have been used in the literature. The model is then applied to two different topics: the influence of software systems upon scientific work and the way that science impacts the software systems. In the former case, I find that technical choices about how "thick" the software infrastructure is impacts both its flexibility in supporting scientific work and changes the amount of work that must be done by users. In the latter case, I find that changes to the scientific software system place responsibility and control for that

[1] The term "infrastructure" as used in this chapter is distinct from the way that the ESMF team uses it. ESMF labels its coupling functionality as "superstructure" and its utility functionality as "infrastructure." For them, user code is sandwiched between these layers: the framework superstructure invokes the user code, and the user code can draw upon utility functionality provided by the infrastructure. For the purposes of this chapter, both of these would be considered infrastructure that scientists can draw upon to more readily implement models.

change upon different stakeholders. Arising out of these findings is a fundamental tension between the stability of software infrastructure and its capacity to respond to scientific innovation.

5.1 Infrastructure and scientific knowledge production

Infrastructure typically refers to “big, durable, well-functioning systems and services, from railroads and highways to telephone, electric power, and the Internet” (Edwards et al. 2009). Edwards et al. go on to argue that information infrastructure is beginning to acquire the characteristics of such systems: robustness, reliability, and widespread accessibility. From the perspective of one using information infrastructure, it is indispensible to the point that one does not actively think about it, and it has achieved a key property of infrastructure, which “becomes visible upon breakdown” but is “transparent” other times (Star and Ruhleder 1996). Star also reminds us that infrastructure is not totally invisible or pushed into the background, but is always visible to certain people. What is infrastructure for many people may be the object of daily work for others. Plumbing is quite salient for plumbers (Star 1999). In the cases here, we consider both individuals who are part of building the “plumbing” of scientific software, and also the users of the software for whom the goal is to transparently provide certain services through the software.

While robustness, indispensability, and transparency are necessary characteristics of infrastructure, they are not sufficient. There is also a matter of scale. A brilliant innovation in transportation, say the Segway scooter, is not infrastructure if only a limited set of people use it. Infrastructure must have high levels of adoption and use among a relevant set of people, leading it to be learned and integrated into other systems. This is a critical issue for software systems like ESMF, ESG, and NCL. These software systems work readily and transparently for some power users, but are striving and aspiring to reach a broader scientific community to move from being a single system to infrastructure for a large segment of climate science. In Hughes’ model of the growth of “large technical systems,” this is the stage of diffusing to new groups and developing a complete set of functionality to meet certain needs within the scientific community (1987). In Edwards’ terms: “system builders imagine an infrastructure” (2003), an activity that we can see these projects actively engage. ESMF has been deliberate about its infrastructural aspirations, with the team recently helping to establish the Global Interoperability Program which aims to further integrate systems like ESMF with others climate science software functionality to facilitate complete services for climate scientists. Similarly, ESG is seeking to integrate post-processing tools such as NCL and CDAT to provide a more complete set of functionality for working with climate science data. Software team members from all three projects participate (to different degrees) in the Global Organization for Earth System Science Portals (GO-ESSP). GO-ESSP aims to coordinate efforts to standardize and link software components intro frameworks that "provide efficient mechanisms for data discovery, access, and analysis of the data" (GO-ESSP 2011). Participants in this effort are acknowledging that the individual software systems that have been developed for climate scientists are very useful, but are not yet integrated into a seamless infrastructure that scientists can use to get their work done. Simultaneously, these integrative projects are an effort for individual software teams to spread usage of their system so that they gain a critical level of usage to establish themselves as the principal means to achieve certain functionality. When software teams speak of aspirational visions of their systems as infrastructure it does more than help them think about the core functionality that they wish to provide and how they need to expand to do so. Infrastructural aspirations also help to persuade others to perceive them as infrastructure, and to feel like they can rely on the software. For instance, rhetoric about stability, support, and long-term commitment are infrastructural ways of speaking that help persuade users that it is worthwhile to begin relying on the software system.

In 2007, the workshop on "The History and Theory of Infrastructure: Distilling Lessons for Scientific Cyberinfrastructure" convened to study lessons from the evolution of infrastructures like railroads and electricity, as the National Science Foundation sought productive ways to promote the design and construction of what they had started calling "cyberinfrastructure" to support scientific work. In short, what does it look like to have effective infrastructure for scientific knowledge production. The *Understanding Infrastructure* report that was the product of the workshop concluded that infrastructure can be grown and nurtured, but is not readily "built" in a top-down fashion (Edwards et al. 2007). Instead, it operates at multiple scales and must be developed, worked into governance, accepted as normal, and appropriated by individuals. Infrastructures already exist to support the creation of scientific knowledge. Edwards et al. cite the creation of statistics to handle increasing amounts of data that scientists wished to assemble as evidence and the evolution of pairwise letter-writing between scientists into collections of papers in journals (Edwards et al. 2007).

Instruments and laboratories are an example of another type of infrastructure for a scientific field. Such technical and social innovations are so essential to science, that they are usefully paired as "technoscience" (Latour 1987; Sismondo 2004). Certain scientific discoveries and even scientific knowledge are not possible without supporting technical apparatuses. Latour sees technology as essential for scaling up the work of scientists, quoting Pasteur:

> As soon as the physicist and chemist leave their laboratories, as soon as the naturalist abandons his travels and collection, they become incapable of the slightest discovery…Outside their laboratories, the physicists and chemists are unarmed soldiers in the battlefield. (Latour 1987)

Technology permits advanced science, and science creates demand for advanced technology. Some disciplines also yield the breakthroughs that feed into advanced technologies. Ribes and Finholt point out that a variety of fields have recognized the importance of considering both social and technical aspects of a system, framed as "sociotechnical" systems or as systems that have both "hard (i.e., technical)" and "soft (i.e., organizational and human)" components (Ribes and Finholt 2009). Technoscience simply applies these ideas to illustrate the blurring between technical development and the expansion of scientific knowledge. Latour argues that scientific and technical work is indistinguishable until labeled as such by human institutions (Latour 1987). Both contribute to the production of scientific knowledge, and both aspects co-evolve together.

In 2003, the NSF's Blue-Ribbon Advisory Panel on Cyberinfrastructure set out a vision for spurring activity on the work that is typically labeled as more technical, and software systems in particular. The report recognizes the way that software projects both benefit science and are spurred on by the technical demands expressed by scientists. As the report begins:

> Scientific and engineering research has been crucial in both the *creation* and the advanced *application* of the amazing products of the digital

revolution begun some sixty years ago – a revolution that increasingly undergirds our modern world. (Atkins 2003)

Consider some of the ways that science and software systems co-evolve in the climate science. The concept and practice of model intercomparison efforts only really became feasible as scientists gained enough computing power to run a large number of simulations for a standard set of experiments and to be able to compare the results. Intercomparisons had been done since the early days of climate modeling, but were formalized and conducted at a larger scale as part of the Atmospheric Model Intercomparison Project (AMIP) of the early 1990s. Using some of the expertise and computing resources that had recently been used for studying nuclear weapons and radiation transport (Edwards 2010b), AMIP specified a single experimental scenario to be run by 33 different modeling groups (Gates 1992; Gates 1997). Simulations were completed in 1993, and helped scientists to determine weaknesses or differences among the models that merited exploration. The AMIP experiments were followed by comparisons of more complete earth system models, with the Coupled Model Intercomparison Project beginning in the late 1990s. AMIP, which had compared simulations of the period 1979-1988, CMIP experiments included simulations of future climate conditions. Consequently, they yielded benefits not just for scientists, but provided a range of possible environmental outcomes for policy-makers to consider. In turn, intercomparison efforts were boosted by the need for a comprehensive picture of the state of climate knowledge for the increasing amounts of climate policy activity, such as the IPCC's assessment reports that synthesized the results generated by different modeling groups. With the growth of intercomparison efforts, the need for tools and organizations to support data sharing became increasingly clear. For intercomparison to become commonplace, scientists needed tools to share the data and organizations to coordinate data distribution. ESG has grown to fill the need for a tool, and PCMDI has been one of the main organizations coordinating data distribution. So scientific, organizational, and political factors helped create the demand for earth system model intercomparison, but it was the technology that enabled it to be practical and the development of specific technical tools was driven by scientist's desire to do intercomparisons. Scientists could not feasibly develop analyses and methods for building knowledge from ensemble models without having the capacity to readily produce and move around these data.

The above discussion of intercomparison projects, coordination between labs, and tool development illustrates co-evolution of science and technology at a macro level. Individuals see this happening at micro levels too. Informants in my study observed the co-evolution of software and science on the ground, too, acknowledging the fashion in which scientific and technical developments progress in parallel:

> In terms of the scientific development, the model and NCL have kind of stayed in parallel pretty much in their development. And so, there is a lot of interaction there, and so, things tend to stay pretty much on pace. (Cameron #1)

Important coordination work keeps model development in sync with the development of other tools like post-processors. As we saw in previous chapters, much of this coordination happens through software support, and power users have a large role in ensuring this happens. By ensuring that software keeps up with their needs, power users can keep the software useful for a broader segment of users who have similar needs. As power users from new branches of climate science engage with the software system, it expands and becomes infrastructural for a greater portion of climate scientists.

I am not going to take a stance on whether ESMF, ESG, and NCL should properly be called "infrastructure" or not. Such a claim is not necessary for my purposes here, may not be helpful, and is certainly not simple to do without knowing the future trajectories of the software projects. However, I do believe that each software system serves as "work infrastructure" that shows "infrastructural aspects of information systems" for some set of users (Pipek and Wulf 2009), whether or not it has become ubiquitous infrastructure for the field of climate science. Pipek and Wulf's concept of work infrastructure borrows from the broader literature on infrastructure to apply it to particular systems. So in the remainder of the chapter we will see some of the infrastructural characteristics of scientific software systems, and the tradeoffs that the teams encounter with designing for these infrastructural properties. In particular, we turn to the way that science and technology influence each other.

5.2 Models of infrastructure

In this section I will introduce a basic model of the relationship between "science" and infrastructure," fully acknowledging and expanding on how the two are closely intertwined. I will briefly compare this to three other models that describe activity and growth of infrastructure: the Atkins model of cyberinfrastructure (Atkins 2003), the Star and Ruhleder model of infrastructure (Star and Ruhleder 1994) that was arranged graphically by Florence Millerand (Edwards et al. 2007), and the Pipek and Wulf model of information infrastructure (Pipek and Wulf 2009). This model will then be expanded and complicated throughout the remainder of the chapter to account to help explain the co-evolution of science and technology and the effects different types of change and design choices.

5.2.1 A model of scientific software

By viewing scientific work as technoscience, intimately involving technical systems, we can understand scientific innovations (or changes in the science) to regularly impact software infrastructure. Changes to "technoscience" are sometimes handled by scientists, and sometimes handled by technical personnel (in our cases, the software teams). The location of the change determines who has a responsibility to implement the change and who has control over the change. These changes are essentially shifts in requirements or the creation of new requirements. Some scientific changes involve only parts of code and work that are under a scientist's control. An example of this is a typical NCL script written by a scientist to analyze her data files. Other scientific changes involve user code but also require a change to the software infrastructure. This could be a new model component written with ESMF that mostly consists of user code but also requires that ESMF support a new type of grid for regridding. Yet another scientific change happen entirely in the software infrastructure, and scientists then gain access to it by using an updated version of the software. Examples of this include improving ESMF's efficiency in routing information among processors, or updating NCL to include new tables about how to interpret GRIB data. We can visualize these different domains of responsibility as layers: with a layer of technical work handled by scientists on top, a layer of software produced by scientific software teams in the middle, and hardware and software produced by industry on the bottom. Figure 6 shows this simple model with examples from ESMF, ESG, and NCL. In these cases, for instance, hardware is a base technology and is generally not part of the science infrastructure systems and is not modified directly by the scientists or software teams.

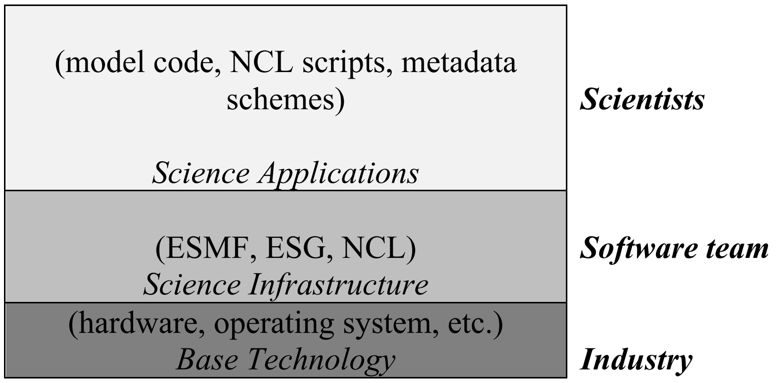


Figure 6: Model of scientific software.

From the perspective of the software projects, which has been the primary viewpoint in this dissertation, the scientists are users and their work involves using technology for science applications. The software team is focused on building science infrastructure, while broader industry produces the base technologies with which science infrastructure is built – including processors, compilers, and operating systems. This model is inspired by the Atkins model of infrastructure (Atkins 2003), which we will discuss more shortly. It is a technology-centered perspective, in that it focuses on the technological components of infrastructure. A model centered on technology is appropriate for the purposes of analyzing software requirements since the requirements that I focus on involve changes to technical systems. This model will help direct our attention to how changes impact the different technical systems used by scientists. At the same time, I fully acknowledge (and have described) the various social and institutional factors that are mediating technical work. Before further developing this model of scientific software we need to contrast it to a few other models put forth in the literature.

5.2.2 The Atkins model: architecture of infrastructure

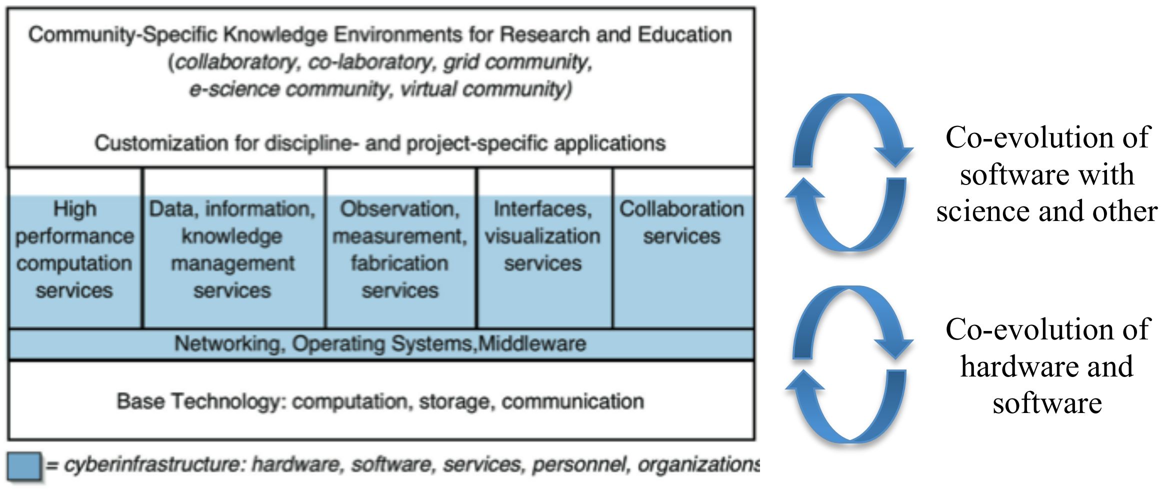

Community-Specific Knowledge Environments for Research and Education
(collaboratory, co-laboratory, grid community, e-science community, virtual community)
Customization for discipline- and project-specific applications
High performance computation services
Data, information, knowledge management services
Observation, measurement, fabrication services
Interfaces, visualization services
Collaboration services
Networking, Operating Systems,Middleware
Base Technology: computation, storage, communication
= cyberinfrastructure: hardware, software, services, personnel, organizations
Co-evolution of software with science and other
Co-evolution of hardware and software

Figure 7: The Atkins model of cyberinfrastructure.

The NSF Blue-Ribbon Panel led by Daniel Atkins produced the model in Figure 7 above. The diagram illustrates different portions of work that happen, highlighting the “cyberinfrastructure” that they concluded was in dire need of cultivation. The figure includes separate sections for different areas of cyberinfrastructure. To place our three cases within this model, ESMF is a “High performance computation service,” ESG is a “Data, information, knowledge management service,” and NCL is an “Interface, visualization service.” This model would play out over time through two main cycles of co-evolution, which are represented to the right of the Atkins figure. First, hardware and software grow in response to each other. Improved hardware requires updates to the cyberinfrastructure, and new ideas in the cyberinfrastructure may demand new hardware. This shows the reliance on base technology, such as processor power, which is driven forward in development by the large resources, demands, and innovation of the computer market more generally. Applications that scientists work on are in the top layer, and similarly co-evolve with the development of cyberinfrastructures. Development of these applications is driven by scientists and is tailored to the needs of a specific group.

This diagram has two main distinctions from the simpler model that I am introducing in this chapter. First, it distinguishes different types of cyberinfrastructure with vertical dividers to emphasize the different realms of activities and services that might be provided (e.g., visualization or computation). In my model, I primarily focus on one type of system at a time and a particular type of infrastructural support for a particular type of scientific activity. However, from a broader perspective of science activity in a field, it is important to recognize the diversity of different types of software infrastructure systems that are needed. In fact, as infrastructure matures we would expect increasing interconnections between the different systems. For the electric infrastructure, for instance, system builders needed to develop electric generators, lights, appliances, and billing systems to track electric usage. Each of these served important uses individually, but it was only when they were all interconnected and agreed on standard voltages and frequencies that the system could work readily, or "infrastructurally." In fact, we are seeing just this effect as there are various points of connection and collaboration between the three software systems I have studied. Furthermore, even within a particular type of scientific activity, vertical dividers are appropriate when there are competing options for users to choose among. For instance, scientists may choose among visualization tools such as CDAT, FERRET, GrADS, and NCL, each produced by a different lab but used more broadly. Until these tools integrate, they are not fully infrastructure but are serving just a subset of the community and so only fulfill a segment of the infrastructural role scientists desire.

The second difference between the Atkins model and mine is that the Atkins model includes a thin layer of cyberinfrastructure classified as "middleware." The middleware layer relays the gains of computing generally into tools that are useful for many different types of cyberinfrastructure. Software such as LINPACK (for linear algebra), Globus (for data transfer), OpenDAP (for selection of subsets of data), and the NetCDF library (for working with NetCDF data files) are all middleware packages used by ESMF, ESG, and NCL. Such "middleware" is important, but I exclude it from my model because there are actually many more layers. For instance, scientific middleware may rely on specialized file systems in supercomputers that could be seen as a layer between middleware and base technology. Or various software systems such as MAPL could lie between ESMF and the scientists code in the application layer. Instead, my simplified model is from the perspective of one software system: what is the base technology that it draws upon, and what science applications do users need to implement themselves when using the software system.

Like my model, the Atkins model depicts the architectural elements of infrastructure, with a focus on technical components as it seeks to promote technical development work. It also acknowledges social aspects to infrastructure development, but leaves that as a layer that bundles "cyberinfrastructure" with personnel and organizational considerations. This technological focus is a fitting basis for my model, which focuses on a technological process: the determination of software requirements.

5.2.3 The Star and Ruhleder model: properties of infrastructure

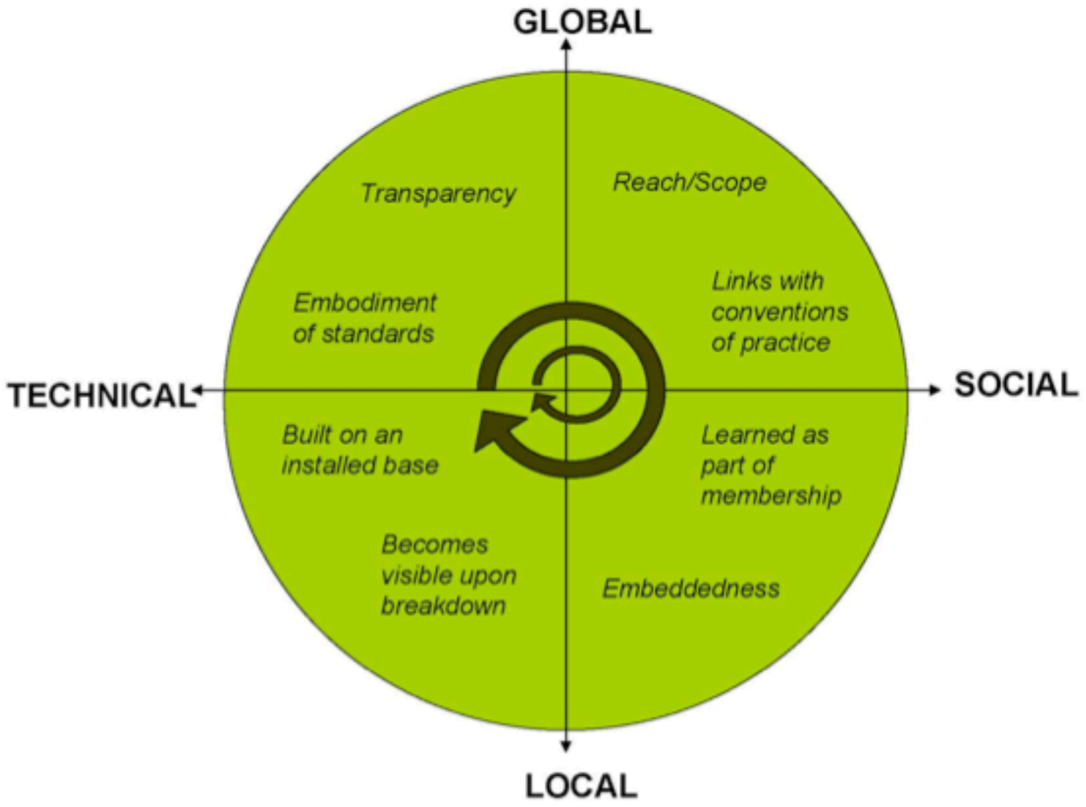

GLOBAL
TECHNICAL
SOCIAL
LOCAL
Transparency
Reach/Scope
Embodiment of standards
Links with conventions of practice
Built on an installed base
Learned as part of membership
Becomes visible upon breakdown
Embeddedness

Figure 8: The Star and Ruhleder model of infrastructure.

Star and Ruhleder organize their conception of infrastructure around properties that emerge from a functioning infrastructure, rather than its architecture (Star and Ruhleder 1996). They fully integrate both social and technical aspects of the infrastructure, and Star later points to the way that infrastructure grows: infrastructure "is fixed in modular increments, not all at once or globally" (Star 1999). Later, Florence Millerand captured this model of infrastructure in a graphic reproduced in Figure 8 (Edwards et al. 2007). This figure includes the properties of infrastructure identified by

Star and Ruhleder, arranged along dimensions of social/technical and global/local, and also adds a cyclical arrow to indicate the way that infrastructure is built up gradually and incrementally. Edwards et al., using this model, argue that infrastructure is best thought about as "growing" rather than being built. This model is not technocentric like the Atkins model, but points out that many types of technical and social, global and local components must co-evolve together as systems coalesce into an infrastructure.

Star and Ruhleder also emphasize that infrastructure is only infrastructure for a particular community, or set of people (1996). One of the main properties of infrastructure is its transparency to this community. For the users, it is simply ready-at-hand and just works. However, for the set of people who are involved in maintenance work on it, it doesn't function as infrastructure at all. Instead, it is the object of their daily work and attention. So depending on who you are and what you are doing, the same system may either serve as infrastructure or an object of work. Each infrastructure thus reaches a certain community for whom it is infrastructure, and is ubiquitous within that community. Infrastructures can then build upon each other.

The properties of infrastructure are important for informing when a systems effort is becoming infrastructural. However, it is less useful for studying the requirements process. The properties that they identify do not have the specificity needed to be requirements, and they do not trace the systems structure in a way that can help to locate how requirements changes affect the system.

5.2.4 The Pipek and Wulf model: growth of infrastructure

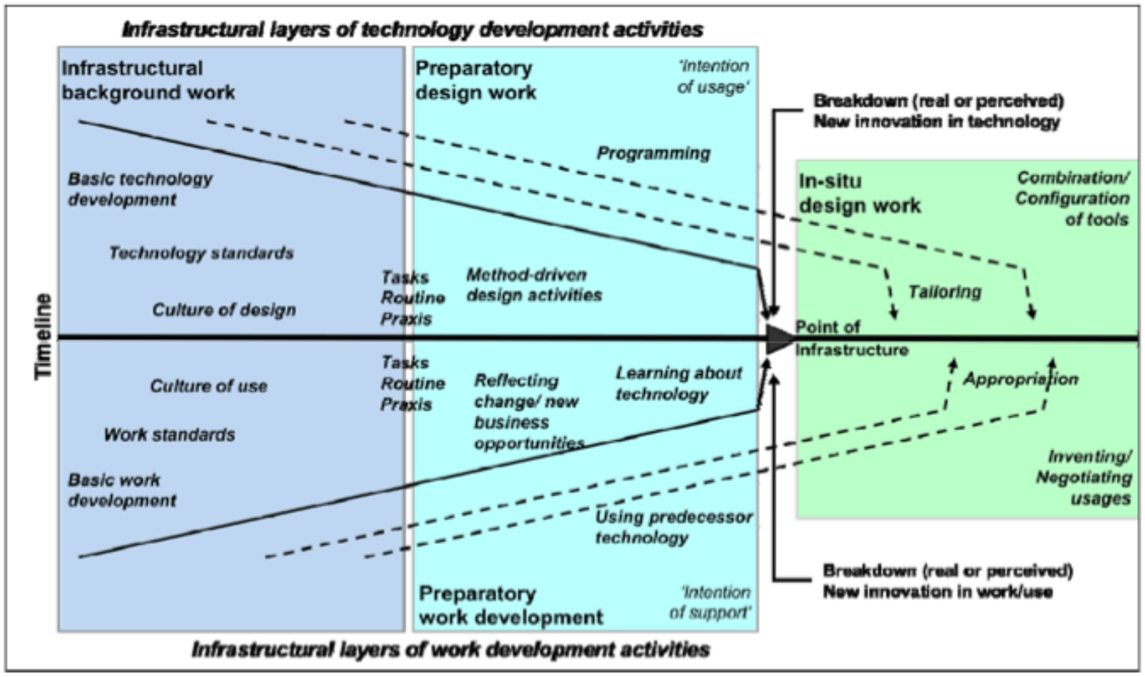

Infrastructural layers of technology development activities
Infrastructural background work
Preparatory design work
'Intention of usage'
Breakdown (real or perceived)
New innovation in technology
Programming
Basic technology development
In-situ design work
Combination/ Configuration of tools
Technology standards
Tasks
Routine
Praxis
Method-driven design activities
Tailoring
Culture of design
Timeline
Point of infrastructure
Culture of use
Tasks
Routine
Praxis
Reflecting change/ new business opportunities
Learning about technology
Appropriation
Work standards
Basic work development
Inventing/ Negotiating usages
Using predecessor technology
Breakdown (real or perceived)
New innovation in work/use
Preparatory work development
'Intention of support'
Infrastructural layers of work development activities

Figure 9: The Pipek and Wulf model of work infrastructure.

Instead of dividing along the lines of social/technical or local/global, Pipek and Wulf produce a model that accounts for two fundamental thrusts in the development of work infrastructure: technology development and work development. Their model (reproduced in Figure 9) is explicit that both of these efforts are vital, and that they need to coevolve together. In describing the model, they highlight that the process described in Figure 9 is repeated again and again in the process of "infrastructuring." The "point of infrastructure" is the moment in which new kludges and adjustments are made to the system, leading it to be better embedded as an infrastructure. This involves both tailoring of the technology, and appropriation of the technology by its users. Thus the system bit by bit, incrementally settles into a role as infrastructure. As we saw in Chapter 3, power users play an especially important role in working with the software team doing technology development. In Pipek and Wulf's terms, power users are the principal drivers of work development activities. By showing how the technology can be appropriated for specific uses within science, and in driving the technology to be tailored appropriately for these uses, they make it easier for like-minded scientists to become users.

Pipek and Wulf's model is helpful in showing the different aspects of infrastructure, and that work needs to happen both in the technology realm and in the realm of the users who are appropriating the technology. In these custom software systems being built to support work practices, there is a close co-evolution between users' work practice and technology developers' products. This corresponds to the model that I am proposing in this chapter (see Figure 6), which architecturally indicates areas that are the responsibility of the software teams (technology development), and areas that are the responsibility of the users (work development). As the infrastructure iteratively grows, changes are often required of both users and of the software team. My model builds on that to highlight the dynamics that exist when making changes to the system, and we will see how there is not a fixed, static boundary between what must be work development and what must be technology development. Instead, the boundary between this is a design decision with important implications for how the infrastructure will co-evolve and whom it reaches.

5.2.5 Obstacles to co-evolution: reverse salients constrain the system

To illustrate the co-evolution of scientific infrastructure, let us trace the impact of a hardware innovation, such as increasing numbers of processor cores in supercomputers. Climate scientists are eager to take advantage of newer systems that allow them to simulate the earth system at an even higher resolution. Higher resolution simulations offer the possibility of improving physics, removing parameterizations of some variables, accounting for important climate features such as the El Niño oscillation, and improving model accuracy overall. Scientists working on the climate models respond to these changes in the "base technology" (see Figure 6) by porting their applications to the new computers in an effort to run higher resolution simulations. ESMF is engaged in this modeling activity, and the software team regularly profiles models using ESMF on new machines to ensure that the ESMF framework is not creating too much overhead (i.e., greater than 5%) as the processor count increases. After models produce the data, scientists then need to pass the data off to data distribution services like ESG or do post-processing analyses with tools like NCL. The increasingly massive datasets generated by models need to be accommodated by the other tools as the impact of the new hardware percolates down. A similar chain of software stresses occur when a new satellite instrument comes online, producing larger quantities of observational data that then need to be assimilated by models, distributed with data portals, and analyzed with post-processing tools.

The risk is that one component of the system will improve either too quickly or too slowly vis-à-vis the rest of the scientific system. Being out of sync in this way can slow scientific knowledge production to the productivity of the slowest component, or it can make other components fail altogether! As NCL struggles to handle larger datasets, it faces two types of problems. The first type of problem is absolute. Large datasets can simply break some NCL scripts. NCL was written for 32-bit computer architectures, using 32-bit variables to address the elements of an array. With the way that NCL implements addressing, each variable in a dataset can address up to 2 gigabytes of data, which is then a cap for the size of variables (e.g., temperature or humidity) in datasets that NCL can handle. Computers have largely moved to 64-bit architectures, but the limitation still remains within NCL. For users trying to import datasets that have individual variables that exceed 2 gigabytes, the program simply crashes. Users have figured out workarounds by importing only parts of a file at a time (e.g., a limited section of time). However, until the NCL team finishes a major re-write of the software that moves from 32-bit to 64-bit variables, this remains a hard limitation.

The second problem is degradation in NCL *performance* when working with large files. In the early 1990s, many models that were being run for the first AMIP experiments used just a single processor and then the data were processed with software such as NCAR Graphics that also used a single processor. Now, NCL (which was derived from NCAR Graphics) still uses only a single processor, but models generate data with thousands of processors. An analysis or visualization that previously required 30 seconds to complete, may now take 15 minutes. This is a quantitative change that has gradually become more significant, to the point where it can change the way that NCL is used. Mistakes in NCL code become more time consuming since it takes some time to complete the script and discover the error. So users are more inclined to try to get it right the first time. Even coffee or a trip to the restroom requires less than 15 minutes, leading users to multi-task or else waste time while waiting for the NCL script to complete. This type of "soft" limitation can have impacts that are just as real as those from a "hard" limitation. Hughes labels this problem of components that are out of sync with the rest of the system as “reverse salients,” points on an advancing front that lag and hold up the progress of the system as a whole (Hughes 1987). System builders must then devote attention to fix laggard components. In the case of NCL, the “front” consists of the various components of the system that support the production of scientific knowledge. A reverse salient is the same basic idea as that of a bottleneck to system development, although Hughes argues that the metaphor of a reverse salient better captures the “uneven and complex change” incumbent to system development.

Scientists experience these reverse salients in the system as obstacles to scientific knowledge production. The risk is then that a few components will lag because they are seen as less “sexy” and less competitive for resources, or because they do not foresee the need to respond to new hardware and new requirements from other components of the scientific process. If software teams such as NCL or ESG only became aware of larger scientific datasets when users gave them feedback about it, there would be a significant delay before they could implement changes to accommodate these datasets. Instead, power users who particularly care about a software system will alert the developers of needs that they will have in the future. Stabilizers such as the project managers engage in big picture planning for the project, and will try to determine hardware and software trends so that their software team is ready to respond to upcoming needs. Other times, scientists will speak out to warn software teams of changes that are on the way. For instance, Dennis and Loft recently analyzed climate models ported to run on new computers with 100,000 processor (2011). Their goal is to provide sufficient notice to avoid bottlenecks in software capabilities, as these climate modelers warn about the effects of modeling advances:

> The ultimate goal of our refactoring applications for performance is to speed up scientific progress. It does little good to increase the performance of the model, if the tools around it slow down or cease to function. Thus, one must examine the entire scientific workflow for bottlenecks. (Dennis and Loft 2011)

Ribes and Finholt warn of tension between “today’s requirements” and “tomorrow’s users” (2007), or the explicit needs of today vs. abstract future needs. Dennis and Loft are seeking to bridge this divide, and bring the needs of future users into the picture so that important new requirements can be introduced. They know there is a tendency to respond mainly to the needs that users are expressing right now, so they warn that the progress of science is at stake and point to a coming wave of user requests that software teams need to start thinking about and addressing right now. Communicating future needs is an important process that helps keep components in the scientific knowledge production system co-evolving, not just on the micro scale of feedback from users via support tickets, but on the larger scales of major changes that need to be instituted in the software architecture to account for trends in the science.

5.3 Infrastructural thickness: Software drives changes in the science

The grandest visions of cyberinfrastructure and software for scientists see new technology as changing science, "revolutionizing" not just the efficiency with which science is done but enabling new *types* of science (Atkins 2003). Within the three case studies we are examining here, the software has a mixture of impacts on science. In some ways, the software simply speeds up and simplifies existing scientific practices. NCL's utility in more easily analyzing data and generating visualizations is an example of this. In other ways the software is changing the types of science that can be done. For instance, ESG's straightforward system for accessing data allows a much broader range of scientists to utilize climate data. Other efforts, such as ESMF's promotion of modeling component interoperability, hold out the potential to transform scientific practice if adoption and software development continue as desired. The impact of these projects, however, is not merely about the software that they provide. We also see significant impact on the scientists through the support provided to users by the software teams. The impact of support on scientists varies significantly though with the three different support models used by the software teams. This section presents some of the changes in scientific knowledge and practice that can be attributed to software projects. It then argues that the thickness, or scope, of the software system impacts when the software team or users have the freedom, control, and responsibility for making changes.

5.3.1 Software changes scientific practice

All three of the software cases seek to improve the efficiency of scientific work. Increased efficiency means that scientists can complete their work more quickly, freeing up time to be used for further research or for other priorities for the scientist. Some efficiency gains are sufficient to enable entire new types of work, or enable science to be done by new people. These more qualitative shifts in scientific practice I label transformative effects.

Scientific software often improves the efficiency of scientists' work. An NCL user articulated: "So, why do we use NCL? Quite honestly, the ease of use. I mean, the fact that the developers are right here in the building helps a lot. The fact that the package is free and it's available on just about every system that we use, makes it just the obvious choice for us" (Cameron #1). These three reasons are all framed around efficiency – with ease of use and minimal hassle for him and his collaborators to access it. NCL originated as a way to grant scientists easier access to the capabilities of the NCAR Graphics library, saving the inconvenience of working with the FORTRAN library by offering a relatively simple scripting language. So from the beginning, NCL has been oriented around saving time and inconvenience for the scientists using it.

The early interface to NCL was object-oriented, in an attempt to gently push scientists toward more modern programming practices and capabilities. Instead, one of the main conditions that CGD imposed before adopting NCL was that the developers must write interfaces that were more functional in nature and similar to the programming paradigms that scientists already used.

> Once they chose NCL, they realized this object-oriented interface wasn't super user-friendly, it was kind of odd for people that were used to FORTRAN programming, it was a little bit of a different look. It wasn't clear to them, you know, how... The whole object-oriented flavor just wasn't popular [chuckle] with that particular user base. So we decided to create an interface that sort of hid that from the users. (Avi #1)

Demands and momentum of existing scientific practice won out, and the software team had to adjust to meet them. So an attempt for the software system to drive change within the scientific community instead resulted in backlash, and the scientists demanded change in the software. The scientists preferred a tool that would incrementally improve the efficiency of their work, without a major disruption or transformation.

ESMF has long had a vision including both efficiency and transformation. ESMF's infrastructure is oriented around providing implementations of common model management functions, such as time-keeping, input and output to disks, and re-gridding.

By implementing these functions in a robust and flexible way, ESMF offers its users a chance to reuse code and lessen the burden of their development work. At the same time, ESMF's superstructure creates technical compatibility for coupling between model components, with the intention that this will allow new model configurations to be created by easily coupling model components together in novel ways. So the oceans component used in one earth system model could be coupled to the atmosphere of another earth system model, a task that has been very difficult in the past. Or an earth system model could more readily be extended to include a new model component like hydrology, increasing interdisciplinary and inter-institutional work to lead to re-mixed and more comprehensive models.

Several of my informants have pointed out that creating technical interoperability between model components does not mean that it is simple as "plug-and-play" to integrate them into a complete new model, although some sponsors may have imagined this. Instead, technical compatibility between components helps, but there is still significant work in ensuring the scientific compatibility between components and then tuning the model parameters to work together in a valid way that reflects the way that the real earth system works. Additional software layers that guarantee scientific compatibility between components, such as MAPL and NUOPC, are being writing to further facilitate model component reusability. However, it is not yet clear whether these will stay within specific institutional collaborations or whether larger segments of the scientific community will agree on a common system. At this point, more modeling groups are still adopting ESMF, and adoption increases the opportunities for model component exchange in the future. The more transformative possibilities for modeling with ESMF rely on network effects. Network effects are commonly seen in the adoption of systems like fax machines, where there is no benefit for a single person to adopt it and relatively small benefit when there are a small number of fax machines. But as fax machines become common over time, buying a fax machine yields much greater value than an earlier purchase would have, because it now allows you to fax a great number of people. Similarly, as more model components are implemented with ESMF, there are many more coupling possibilities and much greater value for each of these components. This in turn creates even more value and incentive for additional models to adopt ESMF, because of the greater number of coupling configurations and new science that are possible through ESMF.

ESMF holds out the possibility of transformative effects for the community by facilitating coupling, and a few groups are taking advantage of that. However, it is the efficiency gains of ESMF's infrastructure that have been most attractive to users. As CESM adopted ESMF, it provided optional ESMF coupling but relied directly on ESMF for time keeping (CESM 2010). Other groups have been very interested in ESMF for its regridding capabilities. ESMF recently bested SCRIP in generating interpolation weights for regridding, providing further advantages to those who adopt the ESMF infrastructure. Under ESMF's new terms for being "ESMF compliant," using the ESMF infrastructure alone will not qualify. However, modeling groups have been increasingly interested in adopting ESMF's infrastructure to improve the efficiency of their model development. It remains to be seen what more transformative effects might result from ESMF's coupling capabilities.

The most coherent usage of ESMF requires user code that has a modular architecture and so can be readily divided into components that can be coupled with ESMF. This is a deliberate choice that is intended to help promote and expedite logical software architecture within models. However, some legacy model codes are monolithic, with large chunks of FORTRAN undivided into logical functions. Modelers in charge of these codes had more resistance to adopting to ESMF, and some who did skirted around the idealized ESMF architecture. Instead of dividing their model into components that are coupled with ESMF, they wrapped the entire model with ESMF. Other model components could then only couple with the model as a whole rather than the model components. In this way they technically "adopted" ESMF and satisfied managerial pressures, but did not realize any of the potentially transformative benefits of a new, standardized coupling mechanism. In this case, much transformation would require much work on the model code with no immediate scientific benefit.

The ESG is a great example of a case where sufficiently advancing the efficiency of a data distribution system can have transformative effects. ESG's main function so far has been the distribution of model data, and particularly the data from models that participated in CMIP3. In the past, scientists would need to work with an individual at each modeling center to gain access to its data. People who were affiliated with a modeling center had relatively easy and direct access to the data produced there. Similarly, people who had contacts at a modeling center could access the data when they are visiting the site by transferring the data to a portable disk or through authorization to access secure FTP sites. However, it was very laborious to get permission to access data from every modeling center that participated in CMIP3, and then to retrieve those data. An ESG team member describes the situation that scientists faced: “And so if you wanted some of that data, then first you had to kind of know that it existed and secondly you had to you know make friends with the person that knew and they had to go manually get it for you” (John #1). By centralizing and making it more convenient to access CMIP3 data, ESG so lowers the barrier to data access that scientists will choose to do so where in the past it would not have been worth it. In the early stages of the project, the ESG team estimated the number of scientists who use CMIP3 data at around 200. However, after making the data available they had some 4,000 registered users, a substantial increase that is partly attributable to scientists who did not previous utilize these data now beginning to download the data because it was so accessible. This cannot be explained by mere trends in the types of model intercomparisons being done and the demand created by IPCC reports. Staff at modeling centers would not have been able to keep up with requests for special access to their systems or the shipping of portable hard drives at the scale of thousands of users, and would have to turn down some scientists – presumably those with fewer connections or affiliations that are significant to particular modeling centers. By making it easier for modeling centers to provide data to large numbers of users, both data users and providers will more often choose to proceed with sharing data. This has enabled the trend toward increasing research using model intercomparisons.

An incremental improvement in the ease of data access tips the balance within individuals’ decision-making. This is especially true for those who do not have an affiliation with the modeling center from which they wish to retrieve data. Previously accessing that data would have required identifying an individual within the institution who would help provide the data to them. This did happen sometimes, but is now much easier. The ESG system facilitates the process by clarifying an entry point for how to go about accessing the data, and then simplifies the approval process by which the ESG team can grant access to the data resources that are desired by the user.

The development of ESG tracks with the emergence of climate model intercomparisons as a major contributor to knowledge about the climate system. Atmospheric model intercomparisons that focused on weather rather than climate had been conducted since the 1950s. However, efforts to compare climate models were limited until the 1990s due to the greater computational needs of long-term model runs (Gates 1997). Climate model intercomparison efforts rose to prominence in the early 1990s with AMIP at PCMDI. Yet these were still relatively small compared to the CMIP3 effort in the mid 2000s. Climate model intercomparison efforts, both in the 1990s and in the 2000s pushed the limits of computational capabilities. ESG started in the early 2000s in response to the increased need to move data for climate science and to develop the technological capacity to support large data transfers. So ESG was developed around just the right time to support data distribution for CMIP3. Many scientists wanted access to CMIP3 data in order to analyze the data and publish findings relevant for the coming IPCC AR4, completed in 2007. Modeling centers were happy to have their model's data used by others, in part because it justified the importance and need to continue funding their model's development.

As part of justifying the utility of its services, PCMDI tracks publications that use CMIP3 data, which now number 574 (PCMDI 2011). Scientists are finding the data very useful. One of my informants indicated that scientists do not feel ready for CMIP5 because there is so much value still to be gleaned from analyzing the CMIP3 data that are available. By enabling access of these data to thousands of users, ESG is helping to transform climate science to include model intercomparison as a key method for scientists to validate models. However, ESG did not spontaneously invent a system that would transform climate science. When the founding ESG team members were considering a focal area for their technology project, they would not have chosen climate model data if there were not already some significant demand for moving and sharing these data as part of the earlier model intercomparison efforts of the 1980s and 1990s. So ESG has already co-evolved with and been party to the increased prominence of model intercomparisons as a means for producing knowledge about the earth system. Recently team members have noticed a new transformation: they attribute their increasingly diverse base of users to the readily available data via ESG as opposed to the old system of data sharing that relied on institutional affiliations and interpersonal relationships. This is certainly heightened by the prominence of the IPCC efforts, drawing scientists from other disciplines and policy makers to the data. This is an opportunity for ESG to transform scientific knowledge in new ways, by facilitating work with neighboring disciplines and by making available data about climate impacts to those who are interested. However, it is also a challenge for the ESG team as they think about whether the data will be understandable and usable for those with less disciplinary training in climate science.

So what are the main takeaways about efficiency gains and transformation of science through software systems? First, attempting to transform too much can meet resistance. We saw this when ESMF tries to enforce specific software architecture choices that would require large changes by modelers. Similarly, NCL had to re-write the interface to its scripting language to be easier for scientists to use than the object-oriented form that programmers found powerful and intuitive. *If the software is too intrusive or demanding than it requires changes to scientists existing work practices and code, and creates resistance for adoption.*

Second, it is difficult to measure increased efficiency of scientific tasks, and perhaps even more difficult to measure the benefits (or costs) of more transformative effects on science. This is not an inherent problem, but does represent a challenge for the software teams who want to measure their impact to advocate for more funding from sponsors. Instead of well-defined metrics counting the time saved by scientists because they can use NCL instead of FERRET, we have rough accounts of the numbers of users. It's difficult for ESG to measure the importance of model intercomparisons to climate science, or to determine how essential ESG is for the process of model intercomparisons. Instead, *user counts are a ready metric, and serve a proxy for the ultimate effects that the software systems enable.*

Third, current users are more interested in efficiency gains to their existing practices than transformative new scientific practices. With ESMF, modelers tend not to be interested in ESMF's coupling unless they are doing a major model re-write or creating a new model anyway. However, modelers *are* interested in ESMF's infrastructure capabilities that allow them to reuse code. Similarly in ESG, those who were doing climate intercomparisons before ESG came about already had a network of personal relationships that allow them to get data from all the different modeling centers. They are only interested in ESG to the extent that it eases their access to those data, which it does not significantly affect. It is the many new users who benefit from the accessibility to the data or the ability to participate in intercomparison efforts. *Transformative effects have been the most interesting to sponsors, and are often emphasized in reports. However, it is the efficiency gains that most excite the early users who are already engaged in the activity that the new piece of software supports.*

5.3.2 Software support changes scientific practice

Time after time, I have heard NCL users praise the support provided by the NCL software team, and how that makes it an easy choice: "What I like most is the support from the NCL team, they are very responsive. I also like the fact that people in my group are using NCL and I can ask them if I have problems." (Josh #1) This user has ready support from the NCL team, and additional support available from the community of users who are his colleagues. Such support makes science more efficient, since scientists can more easily figure out how to complete the tasks they need to do. However, NCL's support has gone beyond technical help, assisting users with the scientific aspects of their inquiries. Some members of the software team help confused users to understand the scientific reasons that their program is not working. For instance, one user was seeing artifacts in the graphics NCL generated not because of confusion about how to use NCL, but rather because the user's data contained blank values. NCL team members suspected this problem based upon the graphical artifacts, and asked to see the data to confirm the issue before explaining that good scientific practice includes manual inspection of the data to check for things like blank values.

It is significant to note here that ESMF, ESG, and NCL have different models for providing support to users, and that these different support models impact the types of benefits that users receive. NCL support casts the widest net, forwarding support request emails to both users and the NCL software team. Correspondence about the problem is usually conducted on the list so that all subscribers can follow. Furthermore, all NCL support emails are publicly archived and indexed by Google. As of February 2011, NCL has about 1100 subscribers to ncl-talk. ESMF support emails go to the entire software team, and are then assigned to a particular member of the team. Although assigned to one team member in particular, the rest of the ESMF team sees all correspondence with the user. ESG support requests go to just one member of the software team, who then handles them or forwards them to other members of the team as needed.[2] Neither ESMF nor ESG support messages are publicly indexed. As summarized in Figure 10, ncl-talk emails go to the broadest set of people, then esmf_support reaches a much narrower set of people, and esg-support goes to just one person. All three support addresses are open for anyone to send to them.

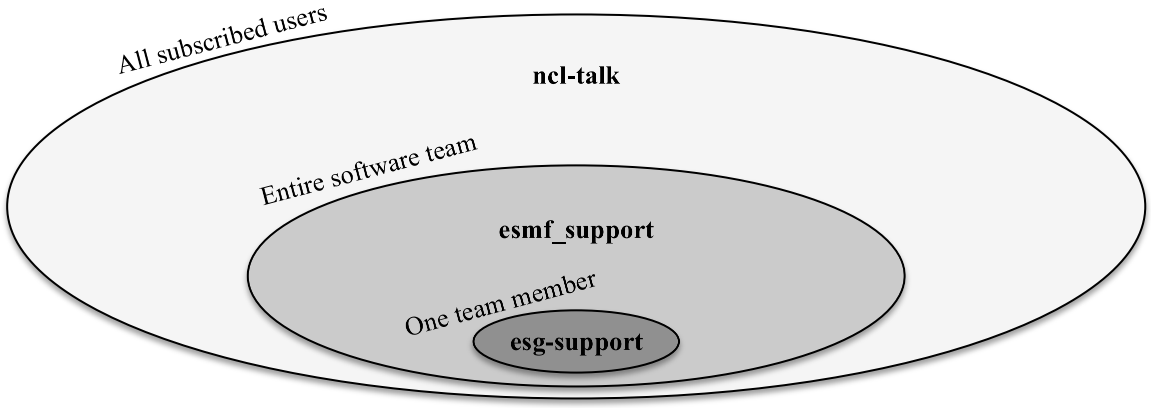


Figure 10: Recipients of support emails vary across projects.

The reason that these different support models are important is that they provide differing capabilities for people to "listen in" on other support conversations, either reading messages in detail or just gaining an awareness of others' problems and possible solutions. In NCL, answers provided by the software team are opportunities to teach the rest of the users as well:

> If the user asks questions, basic questions, we'll answer the question but also tell the user… and then we also tell the community: such questions have examples here. You can check it on the web. So next time the user has a question he can go check there. (Melissa #1)

[2] As I am finishing this dissertation, I learned that ESG is switching to a model similar to that used by ESMF, where support requests will be delivered to a number of people on the software team who will track the request and resolution with a ticketing system. This change is in response to the increased volume of support requests that the ESG anticipates when it begins handling CMIP5 data. My analysis here will be based upon the old/current ESG support model, contrasting it with the models used by NCL and ESMF.

Because the answer is public, it makes it more worthwhile to spend time on the support request since it will be of benefit beyond just one user. The extreme example of this is when an NCL developer uses a support request as an opportunity to both answer the user on ncl-talk and also to create a new case for NCL's web-based library of examples:

> And just yesterday, I spent probably 5 hours writing a script for a user that asked a question, but it was a complicated script that I knew that he wouldn't be able to write. [laughter] And it was one... I try to weigh and I'll say, "Well, is this a question that somebody else is going to answer?" "Is this a popular thing that we get a lot?" And then, if it is, in the case of yesterday, it was, it was a popular thing. I said, "Okay, I'll write this. I'm going to put it out on our examples page and make it available to everybody." So, it's not just, "Oh, one user got 5 hours of my time." (Avi #1)

For ESMF, the support list raises awareness among members of the support team, rather than providing examples for the users. More experienced or managerial members of the software team alert the developer assigned to a support request about whether the user is particularly important strategically for the project. In that case, responses are given particular care and a phone call may be offered to help the user. Team members may also notice when a user is inquiring about a model that they did not know was using ESMF, and they ensure that the model is added to the list of ESMF "users" (models that are at some stage of adoption). At other times, developers add relevant technical detail. For instance, a bug reported by a user is always assigned as the responsibility of one developer, but often one or more additional developers will add useful details based upon their experience with previous similar bugs or when working with a particular section of code.

ESG does not share support inquiries as publicly. Instead, the one designated team member triages support requests – answering or dealing out the support to other team members. Technical questions go to the developers. Requests for an account are handled directly…unless the requester is impatient and rude. Other users and team members, however, would only see these messages if needed. This reduces overhead for most members of the team, but also reduces awareness of the types of requests that are received.

Lurking on the list and listening in to users' support correspondence is a form of *legitimate peripheral participation* (Lave and Wenger 1991). By providing sanctioned ways for less experienced users and team members to unobtrusively participate in support work, these people have an opportunity to learn more about how the software works in a grounded and practical way. They also learn what sort of questions and answers are appropriate or normal in the software support system. In NCL, the software team is explicitly aware of these effects, deliberately speaking to the larger community through an answer to a single user. Similarly, much of the ESMF team's email communication is meant to be overheard – both on their esmf_support email list and also on the esmf_core list used internally by the software team. ESMF uses its support list to apprentice its team members, both passing expertise and portraying a normal mode of interaction with users. Similarly, NCL's support list apprentices users (and members of the software team). ESG's support mechanisms do not provide the same sort of peripheral participation.

For NCL, the ncl-talk list provides some degree of community by traditional definitions of the word, involving shared identity or mutual care. By volunteering to respond to support inquiries, users are demonstrating some degree of "affection" or care for a fellow NCL user. Perhaps the user considers it an opportunity to give back to the NCL community after being helped with an NCL problem in the past. This suggests that the NCL "community" is not purely the constituency that Ribes and Finholt identified in their study of the WATERS project (Ribes and Finholt 2008). It signals a project less concerned about satisfying particular people in order to survive, and instead indicates an active user community that develops tools and scripts that build upon the base NCL software and provides complementary help to fellow users.

By legitimizing and providing a simple venue for many users to participate in NCL support, NCL provides significantly greater impact in scientific training to users. When an NCL developer spends several hours diagnosing a user's problem, and then replies to the list that the user is seeing "ringing" in their visualization because of missing values in the data, then the developer can help train users to inspect their data before trying to analyze or visualize it. In this way, NCL functions not just as a software package, but as a means for building a community. Users can respond to other users, and scientists can share knowledge about how to solve common problems, beyond merely how to use the features of a software package. Rather than individuals working with FORTRAN code to generate visualizations with NCAR Graphics (the state of things 20 years ago), users have both a tool and resources that can help. *The software project can construct web and email resources so that they help with the coding and scientific work that is the responsibility of scientist users. In this way, the project can exert a helpful influence beyond just the software itself into scientific practice relevant to using the software.*

5.3.3 Infrastructural influence and the model of scientific software

Teams building scientific software infrastructure help facilitate scientific work using the software. The technical architecture of scientific software separates work that is the responsibility of users from that of users. However, as we have seen, software teams for projects like NCL both help users to work out their scientific problems and cultivate a space where users can assist each other. Figure 11 illustrates the way in which the activities engaged in by the software team support not just the development of science infrastructure, but also help with scientific work and promote cooperation among users.

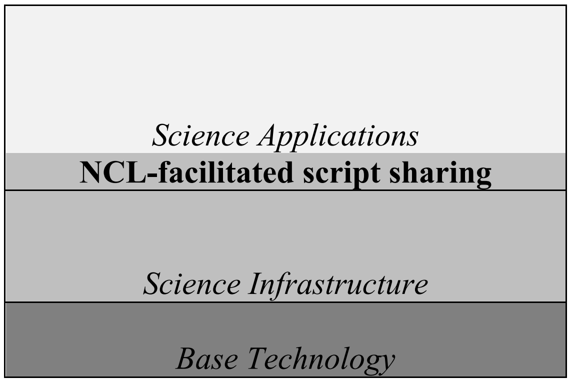


Figure 11: NCL helps users with science and promotes user cooperation.

By promoting cooperation and sharing of code among users, the software project can help to distribute scientific change. When scientists develop an improvement to their own scientific code, initially it stays just with them. As the change is proven valuable, it may then be shared to benefit other users. When the infrastructure cultivates a user community, which NCL does especially well out of the cases considered here, it helps to facilitate the exchange of these science innovations. In a sense, the science infrastructure extends into the science application layer by affecting the way that user code is used and shared. This is just what NCL does by facilitating the sharing of user scripts through ncl-talk and, for the best ones, through the NCL website. Importantly, all of this change can be done readily at the user's initiative without depending on the software team to implement changes. Infrastructure-facilitated sharing is a significant aspect of current efforts in climate science that seek to provide governance mechanisms for community contributions (Edwards 2010a) or that are building workflow systems so that scientists can reuse the set of steps that other scientists use to process data to produce results. My data reveal both that NCL's support structure is unique in building community resources for users, and that NCL support is perceived as a great asset to the software. The team brings this about by stepping out to influence the user's space, helping users with their code and providing a way for users to share resources and knowledge about using NCL.

5.3.4 Infrastructural thickness and the model of scientific software

The value provided by infrastructures influencing the user space foreshadows a larger design about the division of responsibility between the software team and the users. Science changes that are further down, involving more science infrastructure, are both harder to complete and take longer due to the stability of infrastructure. Science changes that are higher up, in the application layer, are easier for scientists to spontaneously change, unencumbered by the infrastructure team's processes of setting development priorities or of making sure all changes are rigorously and correctly implemented. *When a software team takes responsibility for a set of functionality, it becomes both more rigid and reliable, and also becomes less flexible for scientists to tailor to their needs.*

For particularly successful pieces of user code, the infrastructure may expand to incorporate the user code and to make it more robust and available to others. In the early days of ESMF, GEOS-5 was starting to adopt ESMF and needed functionality faster than the ESMF software team provide it. At times, the GEOS-5 modelers would implement a function that they needed in a quick, simple way that would meet their needs. Later, when ESMF developers got around to implementing the functionality in the infrastructure, they would then reference GEOS-5's implementation as they created a more robust version to be included in the infrastructure. In this way the infrastructure expands to include selected user code that is deemed important, incorporating and supporting the functionality for all users. Otherwise users would be responsible for finding or (re)creating the code that would implement the change they desired for their scientific work. Significantly, users are generally not interested in providing support for code they share, but when the code is moved into an infrastructure it is then backed by a team dedicated to supporting the software. Moving the feature into the infrastructure also serves to standardize it. When the calculation for the Madden-Julian Oscillation was first invented, it was implemented by different people differently within NCL scripts. Results varied widely. By creating a standard function within the NCL code to calculate the Madden-Julian Oscillation, then scientists no longer need to replicate that equation and NCL ensures that all scientists using NCL will arrive at the same results. Figure 12 illustrates the expansion of the science infrastructure into an area that was previously the responsibility of users as part of their science applications. This can either be accomplished either by directly referencing or porting the users' code, or by generally expanding the infrastructure's capabilities to include work that users had been doing, especially code for common tasks among users.

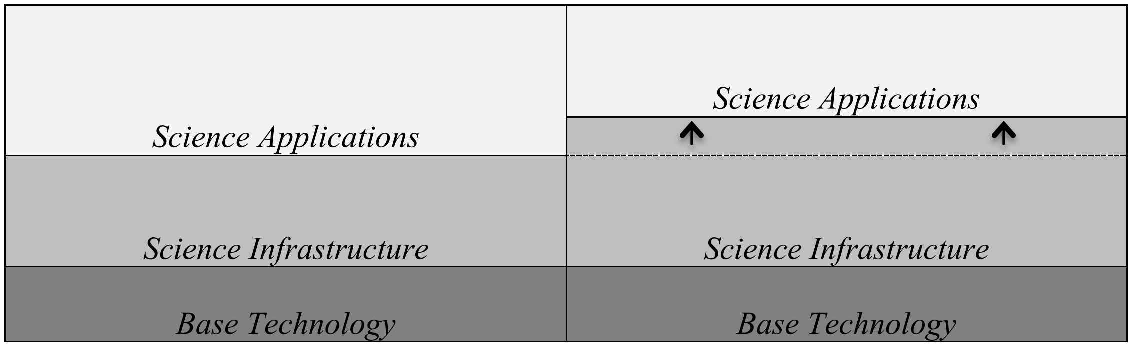


Figure 12: Adding user code into the software infrastructure increases its scope.

This leads us to another observation about the scope of software, a comparison of thick and thin science infrastructure, as seen in Figure 13 (the user "Karen" further below). Teams that are producing science infrastructure face a strategic decision about how "thick" the infrastructure should be. Thicker infrastructure seeks to include more aspects of the scientific work, whereas thinner infrastructure is minimalist and lightweight providing a more limited tool for scientists. Software teams seeking to build a thick, comprehensive infrastructure for scientists will of course face the typical challenges of a large software project. But in particular, they will need to grapple with the fact that more of the changes in science will fall in the region addressed by infrastructure, and hence will require changes in the infrastructure, which we have already seen can be hard or slow to do. Alternatively, thin infrastructure is lightweight and more straightforward in its use, and provides the software team with a narrower focus to their work – knowing what they need to do so that they can do it well. However, there is significant risk in thin infrastructure too. Changes in the science will predominantly fall in the user's application code, requiring changes by users for each new change in the science. Furthermore, science applications are not typically shared (or even sharable), so this work needs to be repeated by each scientist wishing to benefit from a new idea or technique. So thinner infrastructure limits the potential for code reuse.

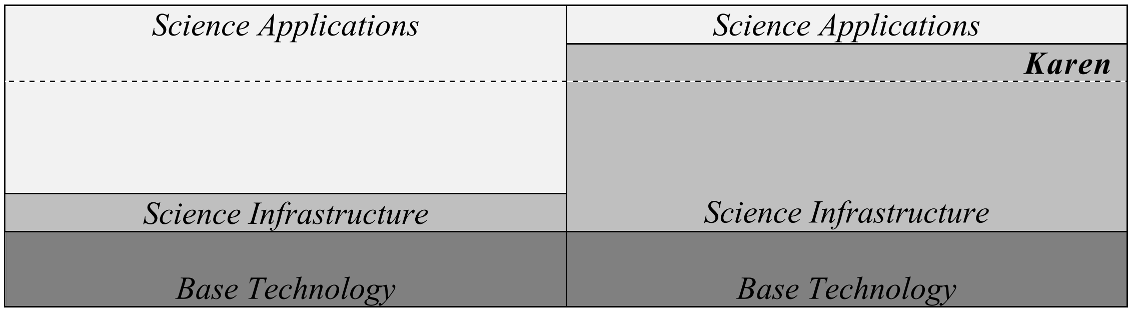


Figure 13: Thick science infrastructure supports novices at the cost of flexibility.

Scientists are concerned about the flexibility of the infrastructure they employ, and this distinction between thick and thin infrastructure helps to explain the reluctance of CCSM modelers to adopt ESMF. As one CCSM modeler expressed:

> So, MCT has an attribute vector but it's just a FORTRAN way of passing data around, really. Whereas ESMF states are more involved, you know, there's multiple layers within that and so forth and they're a complicated derived type and so forth. Whereas the MCT attribute vectors, I don't think are that complicated. So, I think really there is a nice simplicity to MCT that makes it more desirable. (Cameron #1)

Perceptions matter. Irrespective of whether ESMF actually is substantially more work than MCT to use as a coupler, it is seen as a heavyweight and complex and hence less desirable. This may be partly because of a learning curve the users have not yet climbed. However, the complexity also makes it more difficult to tweak the model for new ideas among the scientists, because they need to deal with ESMF or contact the ESMF team, rather than having a minimally intrusive toolkit that allows them to readily make changes to the software for their new ideas. These modelers are interested in the ability to innovate within their own model, and are not particularly concerned with whether it will be capable of running on all the machines that ESMF supports, or whether it will reach all of the ESMF users.

One final insight from Figure 13 can be gained by assessing the effect of infrastructure thickness on a user named “Karen.” Karen may be a graduate student, or ecologist, or a policy-maker, but she only has a limited ability to do scientific work for the particular type of problem that the infrastructure addresses. The distance from the top of the figure to the dashed line represents the limited amount of science application code that she is capable of generating. Thicker science infrastructure enables her to participate in this particular type of scientific work, whereas if more work on science applications were required she would not be able to do it. An example of this is seen with ESG. Previously people like Karen found the barriers to accessing data from earth system models to be too high, but now the infrastructure takes care of a number of technical steps so that Karen is able to go and acquire data. Of course, this represents only one piece of being able to successfully and accurately utilize the data, ESG only addresses data distribution infrastructure in this case. So Karen still needs to figure out how to analyze the data, perhaps using NCL, and would face a new set of infrastructure and work that she needs to do for analysis. In this sense, my model is only addressing a slice of the infrastructure that Karen needs in order to use the data, and post-processing tools may then limit her ability to use the data, rather than being limited by access to the data. A sense of the larger infrastructure, such as in the Atkins model, would incorporate different types of scientific work and could expose some of the tools that are the reverse salients limiting scientific knowledge production.

One risk to providing thick infrastructure that makes data access easy for people like Karen is that she may use the data inappropriately. Previously, Karen may have needed to work with a climate scientist in order to gain access to data, and that scientist would then have the opportunity to provide some instruction and clarification about the data. In this way, too much infrastructure could grant access to scientific tools and data more directly, in a way that squeezes out scientists from having a role in helping to interpret science. This is precisely why the ESG team is concerned about thousands of new ESG users who may use the data in unforeseen or inappropriate ways, and the team is trying to figure out how ESG can help facilitate appropriate usage and understanding of the data.

Pipek and Wulf's model agrees with the joint responsibility of users and the software team for building a fully working system that can produce scientific knowledge. They point out that users appropriate the technology, and that we need to "acknowledge the contributions from designers of information technology as well as from innovators of technology-enhanced practices" (Pipek and Wulf 2009). That is, important work happens both by the software team and by the users. My data concur, and suggest that an *important design decision involves determining the project's ambitions and shifting responsibility and control for work between the software infrastructure and the user's applications.*

5.4 Locating change: Science drives changes in the software

In Chapter 4 we saw the way that software support is critical for updating and elaborating requirements for the software. Here we turn to three types of scientific changes that drive changes in the software: 1) scientific changes driven by hardware improvements; 2) changes due to new scientific methods; and 3) changes due to the software being applied to new problems and different fields of science. I then fit scientific changes into the model of scientific software, locating whether they involve just the scientific applications layer, just the science infrastructure layer, or both. Locating who has control and responsibility for the scientific changes is important, and highlights the effects of having thicker or thinner science infrastructure.

5.4.1 Hardware improvements bring higher resolutions

As hardware improvements have allowed modelers to produce ever-larger quantities of data, software systems like ESMF that support modeling, as well as systems that process and distribute data, must all scale up to handle large datasets. ESMF regularly must be ported and tested on new computer platforms that modelers plan to use. To date, ESG has transferred entire datasets for users to analyze locally. However, anticipating the quantities of data in CMIP5 they have begun re-architecting the system to allow users to download only the subsets of data that they desire (e.g., only certain variables or a specific time/geographic range) and to do analysis remotely on the ESG system so that they only have to transfer the results of the analysis. As discussed before, NCL has also recently struggled to keep up with higher resolution datasets generated by the current generation of climate models. The basic issue is that tools need to co-evolve with the Moore's law-driven improvements in hardware that allow models to produce high-resolution data. Any tool that fails to maintain a capability to work with the data risks becoming a reverse salient that holds back the larger system of scientific knowledge production. Scientists are clear about the importance of handling high resolutions:

> the biggest issue with post-processing using NCL or any package for that matter, is the high-resolution problem. And so, there's a couple of ways to address that. One of them is parallelization and moving towards parallelized post-processing tools is going to be essential for dealing with this. (Cameron #1)

NCL is responding to just this sort of feedback by partnering with Argonne National Laboratories, and using the parallelization knowledge of Argonne's developers to add the capability to utilize multiple processors. To keep up with model data that are computed with thousands or hundreds of thousands of processors, NCL will need to use more than a single processor! The resolution issue looms large within the climate science community, and it is not difficult for each of the software projects to understand that they need to be scaling up to handle increasing amounts of data. However, it does still mean significant work for the software teams, as they need to tap into new skill sets or partner with other software projects. *So the continual innovation in hardware results in changes that software teams need to accommodate, both in the software they develop and also in the composition of the team needed to respond to these changes.*

5.4.2 Scientific innovations among users

Scientists are continually devising innovative approaches to the study of climate, some of which require changes in the software that they use. The changes required of the software range from the big to the small, and could involve new computational techniques, new analyses, or new variables that become important. It could be as simple as an NCL user who was studying super-saturation of water vapor in the atmosphere. However, NCL would only display relative humidity values up to 100%, a seemingly logical limit, until this scientist requested that they re-write it to permit humidity in excess of 100%.

An example of a more complex change for the software systems is the innovation in gridding the earth system that yields many variants of grids, along with the need to regrid data between gridding schemes. Scientists devise novel gridding schemes to maximize the computational capabilities and thus be able to run higher-resolution models. One CGD scientist describes the link between science and technical innovation:

> We're going to cubed spheres in the model. I mean, [it's] all done for... I mean, the ultimate is science, but the problem is that some of the commonly used computational paradigms are not well-suited to these very high resolution grids. So, they come up with new computational stuff. Well, then, that introduces more post-processing stuff because the models just save the data. And so, for example, on some of the cubed spheres, where they intersect you have duplicate points. So, something has to be done to process that. Somebody has to come up with a scheme to handle those types of situations. (Marina #1)

Post-processing tools thus need to respond to the science, or at least the way that the science is operationalized for current computational platforms. Still other times it is not just the computational operationalization that is changing, but new scientific analyses may become important. This scientist describes one change in science over the past decade:

> I think, one thing is maybe that ways of doing analysis may have also changed. So for example in the atmosphere, there's this idea of a Taylor plot, which to me is a strange thing that I don't understand very well, but it's a way of taking multiple variables of interest and you do RMS errors between these variables and obs, and you plot the RMS errors on these sort of correlation wedge plot, and it's a way of quickly doing a quantitative comparison of two model simulations of most of the variables of interest. So, there's maybe 10 quantities on this plot. One of them is the El Niño-Southern Oscillation Index, for example or the amplitude. So, you're going to look at that and you can quickly assess how well a model simulation is doing compared to another. That's an analysis challenge that I think... They came up with this idea and they went to the NCL folks and they said, "How could I plot this?" And it was something that [developer] coded up in a couple of days or something, and it really wasn't too hard... (Cameron #1)

This change in analysis methods was introduced as a response to the increased use of model intercomparisons as a way of understanding the earth system. The developer who coded this new analytic function into NCL needed to have a fairly good understanding of the science, and work closely with the scientist who suggested the new plot. *New techniques and processes that scientists devise need to be accommodated by software systems in order to support an evolving system of scientific knowledge.*

5.4.3 Science expands to new domains

Climate science and findings about climate change have matured now to the point where others are trying to figure out implications for public policy, impacts on businesses and individuals, and interactions with other scientific domains. As more people choose to rely on climate science's findings (despite persistent skeptics) they also need tools that will support the interpretation and utilization of climate science. ESMF and ESG have been particularly mindful of this emerging set of users, as this excerpt from ESMF's 2008 strategic plan reveals:

> Computing and science requirements have evolved. For example, the recognition of global warming as a credible and manifest phenomenon has shifted the nature of the climate science questions emerging from the policy community, national and regional resource management agencies, businesses, and the public. Rather than focusing on establishing scientific consensus on the reality of global warming, current interest focuses on adaptation and mitigation. The models, datasets, methodologies, and research communities involved in these endeavors can be quite different from those involved in traditional global climate modeling. Assessments of regional impacts may require models of specific local-scale processes, such as wildfires and storm surges. Climate or weather models may be used simply as boundary conditions. The local-scale models are often written for Windows platforms instead of supercomputers, in languages other than FORTRAN, and may have products, such as impact maps, that are destined more for analysis by regional managers than by scientists at a national laboratory or university. The modeling infrastructure for these applications must transition easily between desktop systems and high performance computing systems and must enable heterogeneous combinations of data services, models, and visualization and analysis. (Earth System Modeling Framework 2008)

Consequently, one of ESMF's goals is to "Prepare ESMF systems for integration into increasingly heterogeneous computing environments" (ibid). This means both ensuring that ESMF can be used on personal computers and also allowing looser coupling between ESMF climate components and the ESMF components written by scientists studying regional impacts. Similarly, ESG is preparing for an increasingly diverse set of users that include policy makers and scientists in other disciplines. In ESG's case, this means thinking hard about what metadata are needed to help users with different backgrounds to be able to understand the data correctly. The anticipated diversity of users also further validates ESG's push toward server-side analysis and selective download of data, in order to reduce the computational and bandwidth demands put upon users. Policy makers would be less likely to have access to the high-end computing systems that are accessible to climate scientists. NCL is not significantly shifting software features to accommodate new users, but does help teach (through the ncl-talk and the online examples and documentation) new users how to utilize tools like NCL to work with the climate data of interest to them. All three software projects see *value in pursuing new types of users through software features and support*, yet sense a tension between meeting the core needs of the power users and users who bring funding to the project as opposed to the new users who bring greater visibility but not much in terms of funding or valuable feedback.

5.4.4 Locating change in the model of scientific infrastructure

My model of scientific software helps to understand scientific changes such as the three types detailed above. By understanding scientific knowledge production systems as technoscience, or a sociotechnical system, we can see changes to both science and software as important, and equivalent in significant ways. Both represent changes that can improve scientific knowledge, but they involve different people having control over and responsibility for the change. Changes can be seen as a bubble, or region, of the scientific software system that is impacted by the change (see Figure 14). From the perspective of scientific infrastructure (as in this dissertation), when that bubble of scientific change falls over a region that includes the science infrastructure then it results in requirements changes for the software. Changes that fall in the region of science applications are scientific innovations under the responsibility of scientists, who are users of the software infrastructure. If a change to the system of scientific knowledge production can be completed without changing the infrastructure, then scientists can carry out this innovation or advancement in the science by themselves. If the change, at least partially involves the software infrastructure, then the software team in charge of that infrastructure must be involved before the change can be effected.

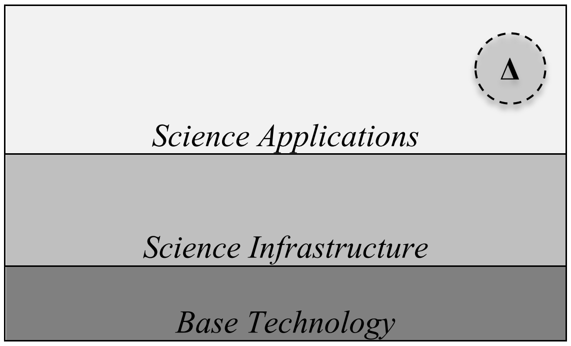


Figure 14: Scientific change, represented by the circle, in the jurisdiction of users.

Consider how this applies to the examples with which I opened this section. First we have the example of users creating and modifying NCL scripts for the purpose of analyzing their data. This small change in science is located entirely within “scientific applications” and does not affect the science infrastructure that it relies upon (this is the arrangement represented in Figure 14). The collection of these personal scripts are what James Howison calls “personal software infrastructure” (2011) that serves as a critical base that individual scientists rely upon, but which is not broadly shared between scientists. Personal software infrastructure is built up over time and may be shared among close collaborators. It allows scientists to quickly tweak a few settings to run a script that will do the analyses that are regularly needed and produce the output needed for publications. We saw how NCL’s support list and examples web pages serve to mobilize parts of this personal infrastructure, making it available to other scientists. More commonly, innovations within the personal software infrastructure are only available to the one who created it or her colleagues. An important feature of the personal software infrastructure is that scientists are familiar with how it works and can manipulate it to do tasks quickly or to explore new scientific techniques and theories. It is both their control and responsibility.

Alternatively, some changes primarily involve the science infrastructure, as illustrated in Figure 15. One example of this would be updating NCL’s GRIB tables so that NCL more correctly and fully understands the GRIB files that NCL scripts read. No changes to the science applications (i.e., the user scripts) are required, yet users reap the benefits of more accurate and complete metadata. Scientists also do not have control over when these changes occur, although the software teams do try to be responsive to their users' needs. Changes to the infrastructure are more robustly tested, and when rolled out, offer new functionality to great numbers of people.

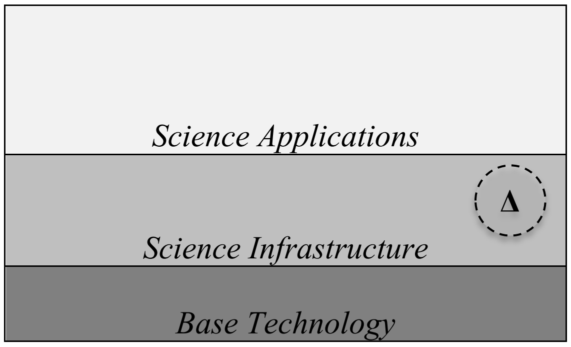


Figure 15: Scientific change in the jurisdiction of the software team.

Some changes may mostly involve work on the science infrastructure, but also require some changes to users' science applications. For instance, when ESMF implemented a new mechanism for efficiently coupling oceans and atmospheres, developers on the software team did most of the work as they worked on the science infrastructure, but users' science applications still needed to makes small changes if they wish to use this new capability. Other times, a change may be primarily within the science application but require a small tweak within the science infrastructure to make it work. More properly, this third type of change is one that straddles science applications and science infrastructure, as shown in Figure 16. A change in the science application also involves a change in the infrastructure, such that new requirements for the infrastructure are needed. This necessitates communication between the two layers, or between users and the software team, in order to work out and understand the new requirements. An example of this is a new model component using ESMF that plans to use a new type of grid, thus requiring that ESMF support regridding between this grid type and other grids. Changes that bridge science applications and infrastructure require coordination between individuals working in both areas, or sometimes an individual will be the bridge who works on both the application and the infrastructure.

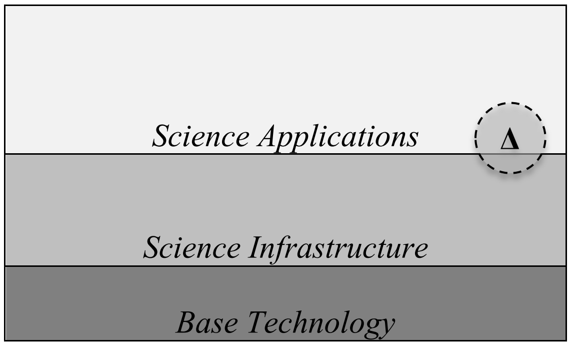


Figure 16: Scientific change in the jurisdiction of both users and software team.

The case of change that occurs in both infrastructure and user applications is common and important. It is an instance of intimate co-evolution, with application and infrastructure changing together to accommodate a new scientific idea (or other change). It also emphasizes the important role of power users. Power users have the knowledge, interest, and relationship with the software team that make them respond to new changes to the infrastructure by adjusting their applications to make use of it.

Determining the location of a given scientific change involves examining who has control over and responsibility for making the change. Changes that are located in the science applications area can simply be done by the users, as they utilize the infrastructure that helps them with science. Changes that are located partially or completely in the science infrastructure constitute requirements changes for the software team, and will involve the team's development cycle. Changes located in the application layer are available to only the individual or perhaps his collaborators. Changes located in the infrastructure layer are available to all users of the infrastructure. Finally, changes that involve the infrastructure layer take more time and effort, as they need to be made robustly functional for a diversity of users and incorporated into the infrastructure's support. *These insights about locating change highlight what is at stake when determining the thickness of an infrastructure, since it affects whether changes tend to fall more within the infrastructure (if the infrastructure is thick) or within the applications (if the infrastructure is thin).*

5.5 Tension between infrastructural stability and scientific innovation

Although the software projects seek to be responsive to new ideas and trends within science, their capacity to respond to scientific innovations is limited and slow (Ribes et al. 2011). The slow response of infrastructure to change in science is partially by design. Infrastructure projects aim to present robust and tested software suitable for use on many different platforms, all of which takes time to develop. Additionally, infrastructure projects serve many different stakeholders and so any potential change needs to be weighed against the priority of other potential changes that stakeholders desire. This tension is due to the very architecture of scientific software systems, as opposed to the tensions identified by Ribes and Finholt, which are related to the context in which scientific software infrastructure is developed (2009).

Software infrastructure systems, or at least the ones in the cases here, are primarily intended to be middleware, supporting the "real" science work done by scientists. But alas, it is not always that simple to tease apart layers of technical and scientific work to allow dynamic scientific activity on top of stout scientific middleware. This issue has been particularly evident for ESMF, which must implement the features and innovations that modeling groups desire. Princeton's Geophysical Fluid Dynamics Laboratory (GFDL) produces one of the major earth system models in the United States, the GFDL Climate Model. The GFDL Climate Model uses the Flexible Modeling System (FMS), a modeling framework that was built in 1997 and served as a precursor to ESMF. FMS helped to establish the basic idea of using a superstructure to couple ocean and atmosphere components, irrespective of the particularities of the scientific code in each component. ESMF was, in part, modeled after FMS and a similar framework based at GSFC. With this experience, representatives from GFDL participated in the very first requirements gathering exercises for ESMF, leading to the initial Requirements Document in 2003. Combined with their experience with FMS, they were intimately familiar with the challenges faced by modeling groups, and the value of the solution proposed through ESMF.

Yet nearly a decade later GFDL has not chosen to adopt ESMF, despite being involved in ESMF since the beginning and having a conceptually similar structure for a model framework. The reasons GFDL has not adopted are instructive. First, there is the matter of the time required to convert the model code. As one of the software engineers at GFDL indicated: “And it's very hard, even for me, to make the case sometimes that you should change all your code. If you start using this bit of software that comes from somewhere else, and you will have... At the end of it, you will have the same features you have today.” (Jennifer #1) It is hard sell to the idea of doing lots of work for no concrete gain. This type of reason can be overcome in the right circumstances – such as when GEOS-5 was undergoing a major code refactor and adopted ESMF along the way, or when BEI was beginning a new model coupling effort and could choose ESMF from the beginning. However, there is a second reason against adoption that is a bigger obstacle, which is a lack of control over the software. This issue is alluded to in the above quote with the phrase “software that comes *from somewhere else*.” Jennifer goes on to speak more of the challenge resulting from increased dependency on outside groups, and the limited responsiveness that GFDL could expect from a group like ESMF:

> I don't think it's possible for an organization or a framework as large as ESMF to be as responsive to users as I have to be. I think there's a kind of ideal size beyond which you can't do that... You cannot be that responsive. I think for ESMF, it's much better to do what they're doing, which is to have an extremely rigorous testing procedure and run it constantly all the time. … The degree of responsiveness I have to the scientific users of the FMS system, I don't think can be replicated by ESMF, because their user base is much more diverse, and it's bigger and more dispersed. (Jennifer #1)

For her, this precludes adoption of ESMF. A large software framework cannot readily respond to the needs of the scientists at GFDL, and her aim is to support their scientific work. By maintaining FMS as an alternate modeling framework tailored for GFDL, her developers can readily tailor it for the scientists' latest ideas, without using extremely rigorous testing and without ensuring that it runs on 30 different computing platforms. Instead of trying to work out scientific innovations within ESMF, she argues that innovation should be done by others and then, once proven, ported into ESMF:

> …you don't innovate the standard. Innovation is done by small groups working without the benefit of standards and then once those methods have been validated then you try to incorporate them into standards. (Jennifer #1)

ESMF has taken just this approach to incorporating innovations from modeling groups. In 2005, GFDL began using a new method for coupling oceans and atmospheres that was much faster than previous methods and allowed them to couple the ocean and atmosphere fields every few minutes, rather than just once or twice per day. This feature, known as the Exchange Grid, is considered one of the most innovative aspects of the GFDL model, is of great interest to other modeling groups (Balaji et al. 2005). Consequently, ESMF decided to implement it so that models that have adopted ESMF may more readily employ this coupling method. However, it was 2010, some five years after GFDL started using Exchange Grids, before ESMF had partially implemented the feature and released it in a beta. The modelers at GFDL are not at all interested in waiting that long, much preferring the way that they had control over the code in FMS to rapidly make the change. The process of importing successful research innovations is important for ESMF and the larger community, but the slow response inherent to the competing priorities and work of creating a robust, general implementation mean that infrastructure like ESMF cannot immediately respond to the new ideas of scientists. If a user wants a change, but it will take even six months, then the user may just do it herself. So my informants see ESMF as appropriately responsive to new model development thrusts, but acknowledge an inevitable time lag before it will implement the new features that are perceived to hold the most value for ESMF's users. Further, this importing of innovations is important because it distributes scientific advances to the many users of the infrastructure. However, GFDL chooses not to use ESMF in part because the infrastructure is "thick" and it would limit its capacity to make scientific changes that involve the coupling framework. For GFDL, this intertwining made it impractical to adopt a modeling framework like ESMF that would limit their ability to advance science by providing a fixed infrastructure.

The case of the Exchange Grid illustrates the challenge for a software infrastructure project to include recent scientific innovations in a robust and general implementation. However, there are areas in our software cases where software infrastructure teams still pursue scientific innovation in limited ways, often in partnership with scientists or outside groups. ESMF has worked to determine which scientific innovations in gridding schemes that it should support, and how to go about doing so.

Initially, ESMF chose to focus on implementing coupling rather than utilities like regridding functions. “The weakness [of deciding to do coupling first] was the fact that we couldn't support all the different kinds of grids that people were using and the innovation in grids has just been really a lot in the last 5 or 10 years” (Jennifer #1). Recently, ESMF has hired developers with more experience in handling grids, and has worked with a researcher at NCAR to help advance ESMF’s regridding capabilities. This sort of partnership with researchers and strategic hiring are the most common approaches to acquiring expertise to build new scientific capabilities. The Earth System Curator project has served as a research umbrella for a number of such collaborations for both ESMF and ESG. It has been particularly useful for working with scientists to understand the latest metadata needs, and then to communicate these needs to the ESG and ESMF software teams so that they can make the necessary changes in the software infrastructure to support new the current science.

With ESMF we see how infrastructure can be so stable that it impedes the deployment of scientific innovations, requiring mechanisms for porting in scientific innovations at a later point rather than supporting scientific innovation within the software itself. ESG has experienced a related challenge. Being located within an information technology department that works closely with scientists, the process of scientific innovation and tight funding have driven them to have a lightweight software development methodology. This adaptation has made it more difficult for them to produce software that operates smoothly at a large scale. The ESG managers noticed differences between industry and lab styles of development, and intentionally hired developers from industry to help provide their software engineering process with greater rigor, and to assist technically as ESG scales up to a system that is closer to the size of commercial systems than to those usually seen within scientific research. By deliberately moving toward a more “heavyweight” engineering process the team intends to become more robust and stable, and will likely follow in ESMF’s footsteps to become less responsive to rapid changes.

The example of NCL adds another perspective to the impact of “scientific innovation” on software infrastructure projects. NCL is quite responsive to users, spending a lot of effort in support and accommodating user requests for (mostly small) changes. NCL is structured such that scientists' innovation tends to happen on top of the infrastructure in the application area, as NCL helps enable scientists to try out new analyses. Some special functions developed by users may become important and then are introduced to the core NCL language. NCL is a relatively thin infrastructure, leaving substantial control to users and thus allowing users to develop many scientific innovations without dependencies on the infrastructure software team. As a scripting language, NCL specifically intends to create an environment that cultivates scientific innovation in data post-processing. Yet like the other projects, it has to make decisions about when to take control of a certain function and "freeze" it, rather than let (or make) users do it for themselves.

5.6 Limitations of the model of change in scientific software

There are some clear limitations to this model of change in scientific software. First, it only accounts for one aspect of scientific work at a time – such as modeling, or data sharing. In reality the situation is multidimensional, and involves multiple science infrastructures that are interconnected. Second, the bubble of "changes in the science" is an abstraction to include many types of changes, and does not address the various ways these changes may be received. For instance, is an intended change in science different from the one that is actually realized once it is negotiated with the software team implementing the infrastructure? The "thickness" of the infrastructure is a very crude way of capturing the complicated costs and benefits imposed on scientists using the infrastructure. Third, the area of science applications focuses on work that scientists accomplish through their own software code, but it can include other scientific tasks that they do as well. This conflation makes the tradeoff between software infrastructure and other work more clear. For instance, ESG does not generally displace coding that scientists used to do, but does displace a difficult process of accessing data by mailing hard drives or trying to find the appropriate person to grant access to data. Nonetheless, the model does represent some of the core features of scientific software that serves as infrastructure and how it relates to changes in science, and it nicely illustrates many of the software's effects that have been observed in the cases of ESMF, ESG, and NCL.

5.7 Conclusion: software layers build from robust to flexible

The model of scientific software that I have introduced and applied helps us to understand the effects of infrastructural thickness, and how different types of changes require/allow action by stakeholders responsible for different layers. Thicker infrastructure supports a more fixed scientific practice in a rich way, but thinner infrastructure more easily accommodates scientists who wish to actively change and modify the science. This difference between thick and steady infrastructure and thin and flexible infrastructure helps to explain ESMF's recent shift in emphasis from research models to production models. Thick infrastructure like ESMF is geared more naturally toward the robustness and slow changes intrinsic to a production quality system, so ESMF has increasingly partnered with agencies like NOAA who are interested in operational model forecasting. It is neither easy nor necessarily helpful to compare the thickness of different types of scientific software infrastructure, but it is helpful to think about the decisions of individual software infrastructure projects that make them relatively thicker or thinner than they would otherwise be. I also do not intend to judge what level of "infrastructural thickness" is most appropriate, but rather to note that software teams producing infrastructure should be conscious of whether their plans for the software align with the community's expectations for flexibility, robustness, responsibility and control.

The tradeoff between thin and thick infrastructure corresponds to a fundamental tension between infrastructure's fixed robustness and the ability of applications that use it to change, especially in response to the continued discovery and experimentation done by research-oriented users. Ribes and Finholt identify a number of tensions in building cyberinfrastructure (2009), but they emphasize tensions arising from organizational factors such as team structure, funding, and time scales. This tension is rooted in a technical problem of control and responsibility for code: software infrastructure that is robustly implemented tends to reduce users' flexibility to readily respond to their needs.

The basic model is also readily complicated to add more nuance and meaning. In particular, it is helpful to acknowledge that there are more than three layers, and that each layer can consist of many different systems. In the case of ESMF, we might see that operating systems like Linux are built on top of hardware. Programming languages like

FORTRAN are built on top of operating systems. Mathematics libraries like LINPACK and inter-processor communication libraries like Open MPI are both built on top of FORTRAN. Modeling frameworks like ESMF are built using both LINPACK and Open MPI. More specific climate modeling assistance is built on top of ESMF through frameworks like MAPL, adding scientific compatibility between components to the simpler technical compatibility that ESMF ensures. Utilizing MAPL, the GEOS-5 model has been built. And on top of GEOS-5, individual scientists configure their personal software infrastructure (Howison 2011), the GEOS-5 scripts, for the specific experiments they plan to run. These layers of software are just a sampling of what is needed, even just for climate models, and there would be similar accounts of software layers for data sharing systems or post-processing and visualization software. As we proceed up through the layers of software, the software systems are increasingly specialized and used by fewer people. Users have control over the personal software infrastructure, and so they can readily change it. However, lower layers of software such as FORTRAN are used by thousands or millions of people, and are robust, stable, and slow to change. This is not to belittle the maintenance work that keeps software systems like FORTRAN so stable (Star and Ruhleder 1996), but to rather acknowledge that such layers of software have figured out how to maintain and support the system with everyday work by people

Layers of scientific software structure are akin to sedimentary rock. Over time, and under pressure, loose sediments are compressed into solid rock. Layers at the bottom were deposited long ago, and tons of layers were added on top. Upper layers do not have the weight of others upon them, and so they are loose and unsteady. Similarly, software in the lower layers is used by many, and must stay stable and robust for the layers above that depend upon it. The middle layers become more solid as they gain users, or as “sediment” is layered on top of them. The upper layer of custom, informal software is flexible and even blows with the trends in the air. In this way, software systems are able to attain a level of flexibility for some code while securing robust software for other code. As we have seen in the scientific software systems here, the divisions between software layers is negotiated among stakeholders and significantly affects the way that scientists do their work.

Chapter 6

Conclusions

Overall, we see the importance of informal aspects of the requirements engineering process, such as the work of power users and user support, whose importance is under-recognized in official documents and accounts of the projects. This is not the only account for explaining how the software projects in my case studies work, but I have argued that these mechanisms have important functions within the projects. Certainly more formal mechanisms contribute to the project too: software performance matters, not just user counts; developers' programming work is vital, not just the user support they provide; and users adopt software because their colleagues use it, not just because it is "thick" enough to provide the functions they need given their level of experience. The mechanisms that are explained in this thesis are all important even while they are not the sole explanations for how requirements engineering works in scientific software projects.

Each of the three main chapters addresses a different aspect of requirements engineering. Chapter 3 focuses on *who* is involved in scientific software projects, with a particular focus on the important role of power users in working out the software requirements. Chapter 4 turns to *how* and *when* requirements engineering work happens, identifying user support as a key location for gathering and clarifying software requirements. Chapter 5 turns to address *what* is at stake in requirements engineering decisions, developing a model to demonstrate the effects of different types of changes in the software project and highlighting the tradeoff between the stability and flexibility of software. The account in chapter 5 more richly explains some of the tensions in cyberinfrastructure development that have been previously identified in the literature.

My data describe the growth of scientific infrastructure. User support constitutes vital maintenance work that sustains the software and fuels its expansion as infrastructure. Software product requirements correspond to the tangible, technical artifacts in an infrastructure, while software process requirements (e.g., user support, training users, making a stable software team) correspond to social components and actions in infrastructure. The software teams in my case studies work to build both technical and social aspects of an infrastructure that supports scientific work, and the software projects have come to exhibit a number of the characteristics of infrastructure.

6.1 Theoretical contributions

This study contributes to understandings of requirements engineering within the computer science community, and to research on scientific infrastructure within science and technology studies and computer supported cooperative work. The work is also valuable in that scientific software development is an understudied environment. Research about scientific software projects is relatively rare, and research about teams of software engineers is especially uncommon.

The business requirements, or objectives of scientific software projects, result in requirements for both the software product and for the software production process. My data speak primarily to process requirements that direct the practices that teams adopt to satisfy the project's objectives. These process requirements and resulting practices illustrate requirements engineering work as it occurs generally and the particular ways it occurs within scientific software projects. In the introduction we saw Herbsleb summarize open questions for the field of requirements engineering, such as which stakeholders need to be engaged in requirements work, whether we can predict requirement stability, and what the effects of different communication media are upon requirements communication (2007). This thesis speaks to each of these issues.

6.1.1 Stakeholders

My basic typology of stakeholders identified four main groups of people engaged with the project: sponsors (or "customers," in the framing of Finholt and Birnholtz (2006)), users, software team members, and collaborating/competing projects. In my case studies, sponsors and the software team relied on user counts to measure project success. Within the funding structures that many scientific software projects encounter, software teams do not generate financial revenue and consequently lack the relatively clear signal of value that revenue provides. Yet sponsors allocating resources to different projects still need a means for assessing the aggregate value of the project to the scientific community. Project reports seek to communicate this aggregate value by painting a picture of the benefits individual users accrue through the software, accompanied by an optimistic estimate of the number of users. For grant-based projects that are not sold for money, user counts serve as the currency for success.

Scientists can thus contribute to a software project's ongoing funding by virtue of simply being counted as a user. Users also contribute to the software system's success by giving valuable feedback via user support channels, and a few power users play a particularly important role in this. Power users do not just possess expertise in using the software, but they also develop relationships with members of the software team. This facilitates the mutual exchange of favors between team members and power users. Power users advocate for the software in the scientific community, alert developers to upcoming requirements for the software, test early releases of the software, and sometimes even provide code that can be imported or referenced while building the software system. In turn, developers prioritize the feature requests of power users, provide them with more thorough and prompt support, and publicize the great work that the power user is doing. It is not necessary to have a large number of power users, and too many power users may prove detrimental. The marginal benefit to the software team from additional power users decreases, while each power user still requires special attention from the software team. Of course there are degrees of power users making them difficult to strictly define, but a good approximation of the number of power users as observed in the case studies would be about one power user per member of the software team.

So while high user counts are necessary to sustain funding and signal the software system's impact, a few power users are particularly important for doing the requirements work. These findings about the role of power users (or friendly users) reflects the findings of De Roure and Goble as they reported about their experiences building e-science software in the United Kingdom (2009). De Roure and Goble worked with friendly users by stationing a developer in the same office as the user, increasing the opportunities for interaction and coordination. The groups that I studied did not have that everyday co-presence, but power users did tend to be in the same building or physically nearby. Team members interact with more distant power users during conferences, workshops, or one-on-one meetings – demonstrating the importance of not just the expertise of power users but their close relationship.

In the "Theory of Remote Scientific Collaboration," Olson et al. identify both a list of desired outcomes for e-science projects and a list of factors that can help lead to those successful outcomes (Olson et al. 2008b). What they lack, and the field needs, is a better sense of the intermediate signals that indicate that "success factors" are indeed present, and that we can expect the project to yield positive outcomes. My research about the role of stakeholders, and especially users, suggests that two such signals are high/increasing user counts and the presence of a few power users. Good user counts indicate that the project is active, and that its prospects for continued funding are positive. Continuing funding for a long period of time is important for the software projects to have time to mature and begin integrating with scientific practice. Power users will both help lead the community in effectively using the software, and indicate that the software features are being tailored to the practical and real needs of users.

6.1.2 Requirements for scientific software

Requirements for scientific software are unstable, because the software team aims to respond to scientists' needs. Requirements evolve in all software projects, but scientific users by nature and intent are constantly seeking to innovate and change their practices to make breakthroughs in science and respond to new ideas in the field (Carver 2009a). For this reason, scientific software operates with a particularly dynamic set of software requirements. Furthermore, scientific software infrastructure projects aspire to readily provide a set of functionality to a broad segment of the scientific community rather than for just an individual lab or research group. However, different scientists try different approaches to research problems. Software teams cannot implement all the software requirements that result from these disparate needs. Instead, software infrastructure projects seek to implement “settled” science that is broadly agreed upon within the community, and to then make this standard functionality broadly available so that researchers can focus on new innovations rather than having to work on problems that have already been resolved. So we can expect that requirements for scientific software are generally more unstable than in other domains, and this will be particularly true when trying to set requirements for areas of science that are not well established and agreed upon. For instance, operational scientific work such as governmental weather forecasting systems would tend to have more stable software requirements than research scientific work such as global climate models based at research laboratories. Software infrastructure to support research thus must discern what aspects of the science have been sufficiently settled to be included as a requirement for the infrastructure, and what is still too unstable and should be left to the discretion of scientists.

Scientific software encodes a current understanding of science. Naur helpfully reveals how the work of a software team is to understand the “theory of the program” (1985). The main work of implementing the software involves developing a detailed understanding of what the software needs to do. When the software’s functionality is only tentatively understood because it is an active area of research, then it is quite difficult for the software team to determine what the software needs to do. Developers need to work very closely with particular scientists to understand how the software should work. Such specific attention and collaboration is common within teams of climate modelers, but impractical for software infrastructure teams that aim to support a community as a whole.

Understanding that software development is fundamentally about learning what the software should do helps to place user support work as a high priority. Regular interactions with users through email support lists help developers learn more about users’ needs, and to identify areas of the program that need work through bug fixes, new functionality or better usability. The personal relationships that power users have with the software team provide an opportunity for close interaction and promote the growth of a shared understanding about what the software does. We saw also how different projects employed different models of user support, where an email to “support” may contact just one team member, the entire software team, or both the software team and other users. Broader circulation of user communication had the effect of including more people in the conversation that creates a common understanding of what software should do, and bootstrapping participation of new team members and new users.

Researchers have begun identifying unique characteristics of software engineering for computational science and engineering (Carver 2009a). I found that my case studies exhibited some, but not all of the characteristics identified in the literature. Current work on scientific software development has focused on cases of “end user development” where scientists are both writing and running the code. Instead, my cases were all with infrastructure projects that provided functionality to scientists, but were not written by the scientists themselves. The literature suggests that developers tend to have a scientific or engineering background. This was true of some of the team members in my study, but other team members in each project had robust training in software engineering too. This resulted in teams that use good software development practices such as testing and effectively use developer tools such as makefiles and code source repositories. My findings agree with the suggestion that requirements change frequently because they “must evolve as the domain is better understood” in addition to the usual changes in user needs and environments (ibid.). My cases also had diverse user communities, were multiple decades in duration, and (sometimes) used esoteric programming languages, as the literature suggests we would expect.

My results reveal an additional difference between scientific and commercial software development that has not been noted in the literature. Because of the grant-based funding model for many scientific software projects, counting users and increasing adoption become important for sustained project funding. This funding-model yields a requirement that the software teams measure the use of the system, resulting in software features such as ESMF’s compliance checker that helps to assess users’ level of adoption and ESG’s detailed metrics of data use. This is similar to the way that commercial software products need to price use of systems in order to extract revenue. However, with scientific software projects the sponsors must assess the aggregate value accrued by users rather than having users express that value themselves through purchasing decisions. This role of sponsors in assessing value results in work by the software team to best represent the value of the software and the high number of users that are benefitting from it. Scientific software infrastructure projects span decades in length, a time scale similar to the largest commercial software products such as the Mozilla/Firefox web browser or Microsoft Office. It is then important for the projects to be able to sustain funding across multiple grants, and team members need to constantly portray their value to the scientific community through user counts and exemplary uses of the software.

6.1.3 Changes in software and science

Not all scientific software is equally sensitive to changes in scientific practice. My model of scientific software infrastructure helps illustrate how change – both of software and of scientific knowledge – interacts with the scientific knowledge production system. Changes located in an area of code that is under the control and responsibility of users can be made without interacting with a software team. Other changes require the software team to respond to users' needs. Software infrastructure such as LINPACK is relatively "thin" in the sense that it addresses a narrowly scoped scientific need, and rarely needs to change since that narrow (and mature) aspect of the science has not changed. Software infrastructure like ESMF is relatively "thick" and needs to respond to new grid types and calendaring systems that it seeks to provide to its users. The model of software infrastructure helps us to understand the connection between architectural decisions about the scope of the software with requirements engineering processes.

Discussion of the model of scientific software also contributes to our understanding of infrastructures. "Thick" software layers offer substantial functionality to support scientific work. However, if these thick layers gain the robustness and stability of infrastructure, they are resistant to change, bringing them into tension with the need to adjust the scientific knowledge production system for changes in the science. In this way, thick infrastructure puts infrastructure and its stability into tension with scientific change. Scientific progress intimately involves change and discovery, unlike physical infrastructures where the end goals such as the reliable transport of goods or electricity do not change.

Stability in scientific infrastructure creates a path dependency and commitment to a certain form of scientific knowledge production. Thicker infrastructures offer greater benefits to users, but also take control of the system away from scientists and commit to certain scientific choices that are more difficult to change. Thin infrastructure reduces the benefits to scientists, but leaves more control in their hands. I observed that some projects resolve this tension between stability, flexibility, and functionality by creating multiple layers of software that are increasingly tailored for a specific use. Because fewer people use the more tailored layers of infrastructure, they are more flexible and easier for the users to influence and adjust in response to their changing needs. At the same time, the most established scientific software functionality can be put in a lower layer of software that is used by many scientists and rarely needs to change. This layering of infrastructure is akin to the formation of sedimentary rock, where the lower layers are compressed and hardened by the mass of layers on top that use them, making them stable. Upper layers of sediment are not yet set, and can more easily be adjusted due to the lower levels of pressure and use exerted on them. The metaphor of sedimentary rock can help us understand the stabilization of infrastructure more generally. As parts of an infrastructure are increasingly used by other systems, they solidify into a stable layer of infrastructure. So there is a sense in which the overall infrastructure that supports scientific work is always thickening to better support current scientific practice. Yet at the same time, individual projects make strategic decisions about how thick *their* layer of the infrastructure should be, as they seek to both provide sufficient functionality for their users and also provide enough flexibility for users to modify and tinker with aspects of science that are still active and are not ready to settle into the infrastructure.

6.2 Limitations and future work

6.2.1 Expanding the case studies

Many of the limitations of this dissertation research are due to a limited sample with certain biases. For instance, the contrast between scientific and commercial software engineering is based upon a comparison to findings in the literature about commercial software development rather than original data. Expanding the case studies to include commercial software projects using the same methodology used here would help to ensure that apparent differences are not merely a consequence of differing methodologies. Another concern is the sample bias created by using snowball recruiting that could mask system “refusers” and unhappy users. Such users are less likely to be associated with the project in a visible way, and are less likely to be mentioned by other stakeholders. My participants did include some such users, who use one of the software systems studied here but refused to use the others. However, I expect that this perspective is still underrepresented, and that such users would have interesting views about ways that they feel excluded from the requirements gathering process or feel like user support does not meet their needs. The limited participant observation time spent with climate scientists in this study could be expanded to gain a better understanding of scientists’ negative views of the software. Further use of participant observation in general would also help provide a more realistic perspective of stakeholder practice, avoiding the inaccuracies introduced as participants’ try to recall information during an interview or represent themselves in the project in a certain (favorable) way.

The three case studies are each one or two decades old, and the nature of requirements engineering work likely shifts over the course of a project. Some participants spoke about requirements work during early stages of the project, but the data are much more directed toward current practices. This thesis has emphasized informal requirements gathering that happens through interactions with power users and through user support lists. However, it is likely that during early stages of the project requirements work is more formal, in order to build up a set of basic software functionality and document the project’s direction for sponsors. Furthermore, there is not a large set of users to support and little time to have developed relationships with power users. It would be illuminating to expand this study to include software projects that are just getting started so as to observe the initial requirements engineering work and the way that requirements practices shift over time into the form that was observed in this study. The current set of NSF DataNet projects would be a good example of large science infrastructure projects that could be studied from their early stages.

6.2.2 Investigating adoption

The role of user counts as described in this thesis is primarily based on data from the software teams and just a few sponsors. It would be helpful to interview more sponsors to detail the way that user counts are tracked and to hear more about what they look at when evaluating the impact of scientific software infrastructure projects. The viewpoint of sponsors would be nicely complimented by a more detailed study of scientists' use and adoption of different scientific software packages. A systematic survey of scientists could provide user counts independent of those obtained by the software projects, allowing comparisons. User counts could prove useful to the software teams to learn about the usage of their software, as well as the usage of their competitors' software. It could also give insights into clusters of institutions, research areas, or even the ages of scientists that tend to use a certain type of software. We may be able to see different types of software that are used by the same scientist, and then check to see what sort of collaboration and integration exists among those software packages to see if they are responsive to the usage patterns of users. A thorough survey would provide insight into all the different stakeholders that we have considered in this dissertation, and also motivate more pointed investigations to understand the patterns that are found.

6.2.3 Detailing responses to the infrastructural thickness tension

Across ESMF, ESG, and NCL we could observe some differences in the "thickness" of infrastructure that the software systems sought to create. However, it would be valuable to contrast software packages that are clearly "thicker" and "thinner." The NetCDF utilities, which provide rudimentary operations on data files, would provide a nice contrast to NCL. Similarly, MCT is a "lightweight" coupling infrastructure that provides a clear contrast to ESMF. We would expect the requirements in these thin infrastructures to be more stable since they involve less science and science that is more agreed upon. MCT does not venture to provide the regridding functionality that modelers keep ESMF busy implementing. Including these additional case studies could also help to address the various responses that different software teams have to the tensions around infrastructural thickness. Some projects seem to respond by deliberately avoiding the extra responsibility and unstable requirements of thicker infrastructure by keeping to a tightly focused purpose for the software. Other projects are content to grow thicker, but do so by importing innovations among the user community once the new idea is proven. Still other software teams seek to do the innovation themselves, including scientists with expertise that allows them to grow the functionality of cutting-edge software. Further investigation of the responses taken could map the different types of responses by software teams and deepen our understanding of the reasons and effects of each type of response.

6.3 Implications for policy and practice

This research has implications for sponsors' policies and for scientific software teams' practice. High-quality software and software engineering practices are important because climate scientists need robust and helpful software. Climate change skeptics persist, particularly in the United States, and so it is important that software used in climate science is able to withstand scrutiny similar to that faced by the personnel impacted by the 2009 "climategate" expose of the University of East Anglia's Climate Research Unit. Without prompting, participants regularly brought up the benefit they hope to bring to the problem of climate change, and were concerned about including sufficient transparency and provenance in their project to protect themselves if inspected or exposed like the staff at the Climate Research Unit.

6.3.1 Use professional software engineers for software work

Software teams consisting of career software engineers seek to support users, in contrast to software teams of computer scientists who prioritize research output (Weedman 1998; Lawrence 2006). The simple advice to software teams is to hire people interested in producing useful software rather than research breakthroughs. Sponsors should also rethink their strategy of using computer scientists as developers for scientific software. My findings indicate that such a practice lies in conflict with goals of achieving a software system with infrastructural characteristics. In practice, software teams could look to hire former scientists discontent with the career path or research focus of scientists. If one finds former scientists with software engineering skills and career goals, they can bring a lot of valuable domain knowledge to the software team. Some of these scientists had even been users of the software system. Team members sometimes referred to this as poaching their users and bringing those who had very specific and relevant expertise onto the software team. A number of the developers on the software teams in this study followed this career path.

There is also an important role for interdisciplinary education that provides combines an engineering background with specific domain knowledge. Novel graduate programs are seeking to develop partnerships between traditional sciences such as climate science with information schools or computer science programs. Students trained in this way will have a heightened understanding of the specific scientific content for which they will produce tools, along with a more rigorous understanding of the informatics and software engineering issues needed to build software efficiently.

6.3.2 Standardize and document user counts

In response to my finding about the way user counts are functioning as a metric of project success, sponsors could seek to standardize the measurement of users or make the measurement process more transparent. Currently, software projects just report the number of people or groups who are using their product. Ideally, we could have a standardized adoption metric to facilitate comparison of similar projects. This would be most feasible for comparing two projects that are competitors, such as CDAT and NCL, since they have similar functionality and similar ways that users engage with the software. Comparing the user count of ESMF with NCL, on the other hand, would be difficult and likely unproductive. Sponsors could also conduct surveys of software usage, ensuring a standardized instrument for measuring adoption rates of different software systems. Alternatively, projects could at least be required to report more detail about how they arrive at their user counts, which would help explain the different types of users and reduce the temptation to over-count borderline users. ESMF is doing some of these very things, providing detailed metrics on their website, and adding detail to distinguish between "prototype" and "routine use" and "application" adoption counts and "component" adoption counts. This increased detail provides the sort of information that sponsors can use to more accurately assess how the project is doing.

6.3.3 Value user support

User support is important for the training in a software system and application to a particular context, in addition to the feedback that it provides to developers. Software projects that aspire to be infrastructure for scientists must provide support to users. The software team should plan for this to be time consuming and should recognize it as important. Similarly, sponsors should look to ensure that software teams include this in their project planning. Provision of support could well be a metric for success, signaling that not only are there more users but that the software is being used more, raising new questions among users. NCL's support seemed particular effective with its practice of rapid replies, provision of an open forum for users to learn by listening in on others, a Google-able archive of messages, and the migration of the best support resources onto the NCL help web pages. Of course a buggy, difficult to use system could generate many support requests, so support metrics ought not be the only metric, but they would help measure a vital process.

6.3.4 Promote relationships with power users

Given the important role we have seen power users play in NCL and ESMF, projects that have users who regularly work with a software system should make sure they build relationships with some of their most expert users. Power users, or friendly users, provide invaluable feedback and services for the software project. Sponsors should also inquire with software teams about whether they have developed relationships with a few top users, and consider these connections as vital along with their raw counts of undifferentiated users. Over time, there is a risk that overly tailoring the software for the needs of expert users could make it less usable for novices. However, software teams are better off acknowledging and seeking to mitigate that risk than foregoing the benefits of a close relationship with power users.

6.3.5 Don't innovate the standard

Infrastructure provides reliable functionality to a broad segment of a community. For software infrastructure, this means facilitating the reuse of commonly needed code by implementing a function "once and for all." For scientific software, this process of creating a standard implementation encodes scientific knowledge into a fixed and standard form. Moving code into scientific infrastructure signals that the relevant scientific knowledge is settled. Once the scientific community has stabilized scientific knowledge, that stable knowledge is encoded and further stabilized by being implemented in scientific software infrastructure that is broadly used by the scientific community.

One participant was quick to point out that "you don't innovate the standard." She argued that infrastructure is not the appropriate location for innovation. Rather it is the boring areas of scientific work that are prime candidates for inclusion in infrastructure, shifting responsibility for the software implementation away from researchers onto the software teams that are building scientific infrastructure. Such software teams should not aspire to create overly “thick” infrastructure, even though it would provide more support to scientist. Rather, they should seek to implement the boring and settled aspects of scientific work, freeing up scientists’ time to work on innovative ideas that are not widely accepted. The tension that software teams experience over the thickness of their infrastructures thus requires discernment about when innovative scientific knowledge is sufficiently stable to be encoded and made broadly available to the scientific community. In a real sense, a software team’s decisions to include a requirement do not just recognize that the science is settled, but make the science settled by providing the scientific community with a ready-at-hand and transparent tool that encodes a certain understanding of the science.

Appendix A

Interview Guide

Avoid terms "stakeholder" and "requirement" unless/until they introduce it. Follow up with specific instances, or the last time that xxx happened.

In this interview, I am interested in who has been involved in [[project name]] and why they were involved. I'm also interested in the [[project name]]'s objectives, and how they have been addressed in the course of the project. Finally, I'd be curious to hear if the science has changed in ways that affect the project in the last 10 years or so.

General

What is your background in terms of education, training, and work experience? (Check CV first if possible.)

What projects are you currently involved in? How is your effort divided?

Work practice

When do you usually do your work?

Where do you usually work from? How does that work for you?

Who do you interact with regularly about this project? How and when do you do so? Prompt to consider email, phone, face to face; formal and informal. Do you interact with users?

Stakeholders and project formation

What's your understanding of how [[project name]] came to be? Does the code have a predecessor?

How has the project (development and maintenance) been funded?

How did you come to be involved in [[project name]]? Why did you choose to do so?

What is your role in [[project name]]? Has it changed in the course of your involvement…?

What other types of people are involved in the project? Prompt for when they became involved, and impressions of how and why they did so

Who affects the way that [[project name]] runs as a project and what it does as a software product? Prompt with a few examples of project operation and software features

Who is affected by [[project name]] as a project and as a software product? Prompt to consider the project's existence at a specific institution, with certain funding

Do you know of times that people or groups have left the project team, stopped using the software, or otherwise left the project? When and why?

Does your project require support from other groups and institutions for packages, modules, algorithms, tools, libraries, etc.? How so?

Addressing requirements

What do you see as the goals and objectives of [[project name]]? How are these addressed?

Have you been involved in forming the project objectives? How did that work for you? Who has been involved in it?

Have there been any disagreements over the project objectives? How has that played out?

What would count as "success" for the project? Is this different from the objectives at all? Who defines it?

What would need to happen to get the software from where it is now to the “perfect” state?

If the software were in the “perfect” state, would all the project’s objectives be satisfied? If not, what else might it take to satisfy the project objectives?

For developers: Tell me about the different types of requests/demands brought by different people and teams. Prompt for requirements beyond functionality and performance: What else does this system need to accomplish for people involved? Name specific stakeholders as necessary.

What different types of things might make it not succeed, or just prove to be challenges? What happens if you get the software wrong? What happens if it takes a long time to produce? [Thinking risks here]

What activities does the team engage in, other than building software? Why?

Developers: What is your perspective on the way this software project has been organized? How have the requirements processes worked?

Have you made trade-offs between code performance optimization and code portability? Other trade-offs?

Changing requirements, changing science

Step back for a minute to consider the scientific field as a whole. Has the science changed over the course of the project? How so? Have scientific work practices changed? How so?

Now let’s go back to this project in particular. Do these changes impact the project? How so?

Have the project objectives or needed software features changed in the course of the project? When and why? Prompt: oversight initially? A new stakeholder? Changing world? Time to right a new grant?

How has [[project name]] affected the way that climate science is done? May refer back to the project’s objectives

Has there ever been a time when a part of the software was developed, but never (or little) used? Has there ever been a time when a large section of code was removed completely? Has there ever been something important that the scientists needed, but the software could not do?

Wrap-up

Who has the final word on project decisions?

For scientists: How do you envision your involvement in software development projects as you plan your work, research, and funding? Does it play out the way you envision?

Do you feel like the project has "taken-off" yet? If so, when did that happen and how did you know it? If not, what would tell you that it has happened?

Is there anything now that could make this project become unused or "fail" in some way? What?

Do you feel like the software is "scientist-friendly?" What would it mean to be "scientist-friendly?"

How does this project compare to other projects that you've been on?

Do you have any "lessons learned" from the project or your work on it? Things that you would keep the same if doing the project again? Things that you would change?

Is there anything else you wish to tell me?

Ask for documents used for goals, tracking, meeting minutes.

References

Abran, A. and J. W. Moore (2004). Guide to the software engineering body of knowledge. Los Alamitos, Calif., IEEE Computer Society.

Ad Hoc Committee for Cyberinfrastructure Research Development and Education in the Atmospheric Sciences (2004). Cyberinfrastructure for the Atmospheric Sciences in the 21st Century. Arlington, VA, National Science Foundation.

Anderson, C. (2008). The long tail: Why the future of business is selling less of more, Hyperion Books.

Aranda, J., S. Easterbrook, et al. (2007). Requirements in the wild: How small companies do it. Requirements Engineering Conference, 2007. RE '07. 15th IEEE International.

Atkins, D. E. (2003). Revolutionizing Science and Engineering Through Cyberinfrastructure: Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructures. Arlington, VA, National Science Foundation**:** 1 v. (various pagings).

Balaji, V., J. Anderson, et al. (2005). The Exchange Grid: a mechanism for data exchange between Earth System components on independent grids. Proceedings of Parallel Computational Fluid Dynamics, Washington, DC.

Batcheller, A. L., B. Hilligoss, et al. (2007). Testing the technology: playing games with video conferencing. Proceedings of the SIGCHI conference on Human factors in computing systems. San Jose, California, USA, ACM.

Blatecky, A. and D. Messerschmitt (2005). Planning for Scientific Cyberinfrastructure, National Science Foundation Shared Cyberinfrastructure Directorate.

Boehm, B. W. (1988). "Understanding and Controlling Software Costs." IEEE Transactions on Software Engineering **14**: 1462-1477.

Bos, N., N. S. Shami, et al. (2004). In-group/out-group effects in distributed teams: an experimental simulation. Proceedings of the 2004 ACM conference on Computer supported cooperative work. Chicago, Illinois, USA, ACM**:** 429-436.

Brynjolfsson, E. (2003). The IT Productivity GAP. Optimize.

Carver, J. C. (2007). Third international workshop on Software Engineering for High Performance Computing Applications. 29th International Conf. on Software Engineering, Minneapolis.

Carver, J. C. (2009a). "First International Workshop on Software Engineering for Computational Science & Engineering." Computing in Science & Engineering **11**(2): 7-11.

Carver, J. C. (2009b). The second international workshop on Software Engineering for Computational Science and Engineering. 31st International Conf. on Software Engineering, Vancouver.

CESM. (2010). "Use of an ESMF library and ESMF interfaces." Retrieved 3/26/2011, from http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc/x2665.html.

Clark, H. H. and S. E. Brennan (1991). Grounding in communication. Perspectives on socially shared cognition. L. Resnick, J. Levine and S. Teasley. Washington, D.C., Amerian Psychological Association**:** 127-149.

Cunningham, W. (2001). "Manifesto for Agile Software Development." Retrieved 2/20/2010, from http://agilemanifesto.org/.

Curtis, B., H. Krasner, et al. (1988). "A field study of the software design process for large systems." Commun. ACM **31**(11): 1268-1287.

Damian, D. (2007). "Stakeholders in Global Requirements Engineering: Lessons Learned from Practice." IEEE Software **24**(2): 21-27.

Damian, D., I. Kwan, et al. (2010). Requirements-Driven Collaboration: Leveraging the Invisible Relationships Between Requirements and People. Collaborative Software Enginnering. A. Frinkelstein, J. Grundy, A. v. d. Hoek, I. Mistrik and J. Whitehead, Springer-Verlag.

Damian, D. and D. Zowghi (2003). "Requirements Engineering challenges in multi-site software development organizations." Requirements engineering **8**(3): 149-160.

De Roure, D. and C. Goble (2009). "Software Design for Empowering Scientists." Software, IEEE **26**(1): 88-95.

Dennis, J. M. and R. D. Loft (2011). Refactoring Scientific Applications for Massive Parallelism. Numerical Techniques for Global Atmospheric Models. P. Lauritzen, C. Jablonowski, M. Taylor and R. Nair, Springer.

Earth System Grid (2006). ESG II Final Report: Turning Climate Datasets into Community Resources, U.S. Department of Energy Office of Science.

Earth System Modeling Framework (2008). ESMF Strategic Plan for 2008-2010. Boulder, CO, National Center for Atmospheric Research.

Earth System Modeling Framework (2009). ESMF Annual Report 2009. Boulder, CO, National Oceanic and Atmospheric Administration.

Earth System Modeling Framework (2010). ESMF Project Plan 2010-2015. Boulder, CO, National Oceanic and Atmospheric Administration.

Earth System Modeling Framework. (2011). "About ESMF." Retrieved 2/7/2011, 2011, from http://www.earthsystemmodeling.org/about_us/.

Easterbrook, S. and T. Johns (2009). Engineering the Software for Understanding Climate Change. Computing in Science and Engineering, IEEE Computer Society. **99**.

Edwards, P. N. (2003). Infrastructure and Modernity: Force, Time, and Social Organization in the History of Sociotechnical Systems. Modernity and technology. T. J. Misa, P. Brey and A. Feenberg. Cambridge, Mass., MIT Press**:** vi, 421 p.

Edwards, P. N. (2010a). Scaling Up: Introducing Commoditized Governance into Community Earth Science Modeling. National Science Foundation award #0941386.

Edwards, P. N. (2010b). A Vast Machine: Computer Models, Climate Data, and the Politics of Global Warming. Cambridge, MA, MIT Press.

Edwards, P. N., G. C. Bowker, et al. (2009). "Introduction: An Agenda for Infrastructure Studies." Journal of the Association for Information Systems **10**(5): 364-374.

Edwards, P. N., S. J. Jackson, et al. (2007). Understanding Infrastructure: Dynamics, Tensions, and Design, Report of a Workshop on History & Theory of Infrastructure: Lessons for New Scientific Cyberinfrastructures.

Finholt, T. A. (2002). "Collaboratories." Annual Review of Information Science and Technology (ARIST) **36**: 73-107.

Finholt, T. A. and J. P. Birnholtz (2006). If We Build It, Will They Come? The Cultural Challenges of Cyberinfrastrcture Development. Managing Nano-Bio-Info-Cogno Innovations: Converging Technologies in Society. W. S. Bainbridge and M. C. Roco. Netherlands, Springer**:** 89–101.

Gates, W. L. (1992). "AMIP: The Atmospheric Model Intercomparison Project." Bulletin of the American Meteorological Society **73**(12): 1962-1970.

Gates, W. L. (1997). AMIP: The Atmospheric Model Intercomparison Project. Livermore, CA, Program for Climate Model Diagnosis and Intercomparison, Lawrence Livermore National Laboratory.

Glinz, M. and R. Wieringa (2007). "Guest Editor's Introduction: Stakeholders in Requirements Engineering." IEEE software **24**(2): 18.

GO-ESSP. (2011). "Global Organization for Earth System Science Portals." Retrieved 3/9/2011, from http://go-essp.gfdl.noaa.gov/.

Goncalves, M. K., C. R. B. de Souza, et al. (2009). Initial findings from an observational study of software engineers. Computer Supported Cooperative Work in Design, Santiago, Chile.

Haigh, T. (2001). "Inventing Information Systems: The Systems Men and the Computer, 1950-1968." The Business History Review **75**(1): 15-61.

Herbsleb, J. D. (2007). Global Software Engineering: The Future of Socio-technical Coordination. 2007 Future of Software Engineering, IEEE Computer Society.

Herbsleb, J. D. and A. Mockus (2003). "Formulation and preliminary test of an empirical theory of coordination in software engineering." SIGSOFT Softw. Eng. Notes **28**(5): 138-137.

Herbsleb, J. D., A. Mockus, et al. (2000). Distance, dependencies, and delay in a global collaboration. Proceedings of the 2000 ACM conference on Computer supported cooperative work. Philadelphia, Pennsylvania, United States, ACM.

Hey, T., S. Tansley, et al. (2009). The Fourth Paradigm: Data-Intensive Scientific Discovery. Redmond, WA, Microsoft Research.

Hinds, P. and D. Bailey (2003). "Out of sight, out of sync: Understanding conflict in distributed teams." Organization Science **14**(6): 615-632.

Howison, J. (2011). Scientific software as a socio-technical ecosystem: incentives and interventions. Ann Arbor, MI, Talk at the University of Michigan School of Information on 2/9/2011.

Howison, J. and J. D. Herbsleb (2011). Scientific software production: incentives and collaboration. Proc CSCW, Hangzhou, China, ACM.

Hughes, T. P. (1987). The Evolution of Large Technical Systems. The Social Construction of Technological Systems. W. Bijker, T. Hughes and T. Pinch. Cambridge, MA, MIT Press**:** 51-82.

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. IEEE std 610.12-1990.

Jirotka, M. and J. A. Goguen (1994). Requirements engineering: social and technical issues. London, Academic Press.

Kendall, R., D. Fisher, et al. (2008). "Development of a Weather Forecasting Code: A Case Study." IEEE Software **25**(4): 59-65.

Kling, R. and W. Scacchi (1982). "The Web of Computing: Computing Technology as Social Organization." Advances in Computers **21**: 3-85.

Landauer, T. K. (1995). The Trouble with Computers: Usefulness, Usability, and Productivity. Cambridge, MA, MIT Press.

Latour, B. (1987). Science in action : how to follow scientists and engineers through society. Cambridge, MA, Harvard University Press.

Latour, B. (1993). We Have Never Been Modern. Cambridge, MA, Harvard University Press.

Lave, J. and E. Wenger (1991). Situated Learning: Legitimate Peripheral Participation. Cambridge, Cambridge University Press.

Law, J. (1991). A Sociology of monsters : essays on power, technology, and domination. London ; New York, Routledge.

Lawrence, K. A. (2006). "Walking the Tightrope: The Balancing Acts of a Large e-Research Project." Computer Supported Cooperative Work **15**(4): 385-411.

Lee, C. P., P. Dourish, et al. (2006). The human infrastructure of cyberinfrastructure. Proc CSCW. Banff, Alberta, Canada, ACM.

McConnell, S. (1996). Rapid Development: Taming Wild Software Schedules, Microsoft Press.

Middleton-Link, A., D. Kennison, et al. (1995, July 1995). "NCAR Graphics Fundamentals: Preface." Retrieved 30 August, 2010, from http://ngwww.ucar.edu/fund/matter/preface.html - HDR%203%2026.

Naur, P. (1985). "Programming as theory building." Microprocessing and Microprogramming **15**(5): 253-261.

NCAR Command Language. (2011). "NCL Overview." Retrieved 2/7/2011, 2011, from http://www.ncl.ucar.edu/overview.shtml.

Olson, G. M., A. Zimmerman, et al. (2008a). Scientific collaboration on the Internet. Cambridge, Mass., MIT Press.

Olson, J. S., E. C. Hofer, et al. (2008b). A Theory of Remote Scientific Collaboration. Scientific collaboration on the Internet. G. M. Olson, A. Zimmerman and N. Bos. Cambridge, Mass., MIT Press**:** xii, 406 p.

Overpeck, J. T., G. A. Meehl, et al. (2011). "Climate Data Challenges in the 21st Century." Science **331**(6018): 700-702.

Pachauri, R. K. and A. Reisinger (2007). Fourth Assessment Report: Climate Change 2007: Sythesis Report. Geneva, Switzerland, International Panel on Climate Change.

PCMDI. (2011). "WCRP CMIP3 Subproject Publications." Retrieved 2/26/2011, from http://www-pcmdi.llnl.gov/ipcc/subproject_publications.php.

Perlow, L. A. (1999). "The Time Famine: Toward a Sociology of Work Time." Administrative Science Quarterly **44**(1): 57-81.

Perry, D. E., N. A. Staudenmayer, et al. (1994). "People, organizations, and process improvement." Software, IEEE **11**(4): 36-45.

Pipek, V. and V. Wulf (2009). "Infrastructuring: Toward an Integrated Perspective on the Design and Use of Information Technology." Journal of the Association for Information Systems **10**(5): 447-473.

Ribes, D. and K. S. Baker (2007). Modes of Social Science Engagement in Community Infrastructure Design. Proceedings of Third International Conference on Communities and Technology, Lansing, MI, Springer.

Ribes, D., A. L. Batcheller, et al. (2011). Responses to tensions in infrastructure development: Toward understanding sociotechnical design patterns. Workshop on Changing Dynamics of Scientific Collaboration at HICSS 2011, Manoa, HI.

Ribes, D. and T. Finholt (2009). "The Long Now of Technology Infrastructure: Articulating Tensions in Development." Journal of the Association for Information Systems **10**(5): 375-398.

Ribes, D. and T. A. Finholt (2007). Tensions across the scales: planning infrastructure for the long-term. Proc GROUP. Sanibel Island, Florida, USA, ACM.

Ribes, D. and T. A. Finholt (2008). Representing community: knowing users in the face of changing constituencies. Proceedings of the ACM 2008 conference on Computer supported cooperative work. San Diego, CA, USA, ACM.

Ribes, D. and C. Lee (2010). "Sociotechnical Studies of Cyberinfrastructure and e-Research: Current Themes and Future Trajectories." Computer Supported Cooperative Work (CSCW) **19**(3): 231-244.

Richardson, K., W. Steffen, et al. (2009). Climate Change: Global Risks Challenges and Decisions, Synthesis Report, University of Copenhagen.

Ronkko, K., O. Lindeberg, et al. (2002). 'Bad practice' or 'Bad methods' are software engineering and ethnographic discourses incompatible? International Symposium on Empirical Software Engineering, Nara, Japan.

Sangwan, R., N. Mullick, et al. (2006). Global software development handbook, CRC Press.

Segal, J. (2005). "When Software Engineers Met Research Scientists: A Case Study." Empirical Software Engineering **10**(4): 517-536.

Sismondo, S. (2004). An introduction to science and technology studies. Malden, MA, Blackwell Pub.

Stalder, F. (1997). "Actor-Network-Theory and Communication Networks: Toward Convergence." Retrieved 3/6/2008, 2008, from http://felix.openflows.com/html/Network_Theory.html.

Star, L. (1999). "The Ethnography of Infrastructure." American Behavioral Scientist **43**(3): 377-391.

Star, S. L. and K. Ruhleder (1994). Steps towards an ecology of infrastructure: complex problems in design and access for large-scale collaborative systems. Proceedings of the 1994 ACM conference on Computer supported cooperative work. Chapel Hill, North Carolina, United States, ACM**:** 253-264.

Star, S. L. and K. Ruhleder (1996). "Steps Toward an Ecology of Infrastructure: Design and Access for Large Information Spaces." Information Systems Research **7**(1): 111-134.

Unidata. (2010). "NetCDF (Network Common Data Form)." Retrieved 2/20/2010, from http://www.unidata.ucar.edu/software/netcdf/.

Veinott, E. S., J. Olson, et al. (1999). Video helps remote work: speakers who need to negotiate common ground benefit from seeing each other. Proceedings of the SIGCHI conference on Human factors in computing systems, Pittsburgh, Pennsylvania, United States, ACM Press.

Vessey, I. and A. P. Sravanapudi (1995). "CASE tools as collaborative support technologies." Commun. ACM **38**(1): 83-95.

Walz, D. B., J. J. Elam, et al. (1993). "Inside a software design team: knowledge acquisition, sharing, and integration." Commun. ACM **36**(10): 63-77.

Weedman, J. (1998). "The Structure of Incentive: Design and Client Roles in Application-Oriented Research." Science Technology & Human Values **23**(3): 315-345.

Wiegers, K. E. (2003). Software requirements : practical techniques for gathering and managing requirements throughout the product development cycle. Redmond, Wash., Microsoft Press.

Williams, D. (2009, 3/12/2009). "Earth System Grid Center For Enabling Technologies (ESG-CET) -- ESG." Retrieved 2/7/2011, 2011, from http://esg-pcmdi.llnl.gov/.

Wilson, G. (2006). "Software Carpentry: Getting Scientists to Write Better Code by Making Them More Productive." Computing in Science & Engineering **8**(6): 66-69.

Woolgar, S. (1991). Configuring the user: the case of usability trials. A sociology of monsters: essays on power, technology and domination. J. Law**:** 58-97.

Zimmerman, A. S. and B. A. Nardi (2006). Whither or whether HCI: requirements analysis for multi-sited, multi-user cyberinfrastructures. CHI '06 extended abstracts on Human factors in computing systems, Montreal, ACM Press.