Tensions between Scientific Programming and the Scientific Method

by

DAVID GRAY WIDDER

A THESIS

Presented to the Department of Computer and Information Science
and the Robert D. Clark Honors College
in partial fulfillment of the requirements for the degree of
Bachelor of Science

June 2017

An Abstract of the Thesis of

David Gray Widder for the degree of Bachelor of Arts
in the Department of Computer and Information Science to be taken June 2017

Title: Tensions Between Scientific Programming and the Scientific Method

Approved: _____

Professor Stephen Fickas

This thesis evaluates eleven scientific programmers against six best practices. The six best practices extracted from literature are: 1) the presence of a Distinct Design Phase, 2) Documentation, 3) the Use of Existing, Trustworthy Code, 4) the Use of Formal Version Control, 5) a Testing procedure, and 6) the Public Release of Code. A survey and interview were conducted on eleven participants to produce a case study on each participant's purpose of their code, their self identified process difficulties, their tool use, their self identified areas for growth, and their perceptions of the importance of programming to their discipline. An evaluation of the extent to which the six best practices were followed is then presented. To conclude, three recommendations on how to increase adherence to best practices are presented: education about existing processes and tools, the adaption of existing processes and tools, and incentivizing adherence to best practices.

Acknowledgements

I would like to thank Professors Stephen Fickas, Boyana Norris, and Helen Southworth for serving on my thesis committee and for their advice and support throughout this long but enjoyable process. I would like to extend special thanks to my primary thesis advisor, Stephen Fickas, for all the opportunities he has provided that helped get me interested in research in the first place, and for his support throughout my college career. I would like to thank my CHC advisor, Helen Southworth, for her advice and support over these past four years. I would like to thank Dr. Kathleen Freeman for her expert advice as I weighed different course and career options. I would like to also thank the kind people who agreed to be interviewed as a participant for my thesis. Finally, I would like to thank all of my friends, teammates and colleagues who kept me sane through my four years here. Go Ducks!

Table of Contents

List of Figures

List of Tables

Introduction

Programming is increasingly relied on as a tool of scientific discovery, and therefore we must give the programming process the same scrutiny given to any other scientific process. While the general public may associate the products of code with the apps they use on their smartphone or laptop, many are less aware of less visible uses of code, such as those involved in scientific research. The consequence of sloppy commercial apps may lead to loss of profit or damage to a company's reputation, but the consequence of sloppy scientific code may be the corruption of the scientific process to the detriment of the scientific community (Carver 2007), arguably an outcome with more lasting and pervasive negative effects. Therefore, I argue increased attention should be paid to the use of code in the scientific process, and I offer this thesis as a small contribution to a growing body of literature focused on this topic.

The contributions of my thesis are: 1) a summary of best practices for scientifically valid programming extracted from literature, 2) eleven case studies of scientific programmers and their processes including what their code does, their self identified process difficulties, their tool use, their self identified areas for growth and their perceptions of the importance of programming to their discipline, 3) an evaluation of the manner and extent to which participants conformed to the best practices identified, 4) an overall conclusion including suggestions of how to effectively encourage the use of best practices informed by my case studies.

Background

Software Development Model

There are many varieties of the Software Development Model, each designed to meet different needs. One 2010 ACM SIGSOFT article documented 12 distinct approaches to applying different Software Development Models. (Ruparelia 2010) Most models include some combination of the following: Requirements Analysis, Design, Development, Integration, Testing, Deployment, Maintenance and Evaluation. Models range from the sequential Waterfall method where each stage is completed sequentially for the whole system, to the Agile method where short development intervals incrementally add to a growing system. Different models seek to satisfy different requirements: the Spiral model emphasizes risk reduction and thus is appropriate for safety critical settings, whereas the Agile model emphasizes rapid progress on incremental improvements, and thus may be appropriate for settings where being able to work with stakeholders in a dynamic fashion is of paramount importance. (Ruparelia 2010)

Professional End User Programmers

The term “Professional Programmer” refers to those who write code as their primary job function in order to build packaged software. In contrast, End User Programmers are people who write programs in support of their work, but not as their primary job function. Many End User Programmers use special-purpose languages, such as spreadsheet languages or Visual Basic, but some use regular programming languages, such as Python or C. (Myers 2006) However, the term “Professional End

User Developer" has been used to describe people who operate in technical, knowledge-rich domains, who have little formal training in software engineering and would not describe themselves as software engineers (in common with EUPs) but for whom learning a regular language does not present a problem (in contrast to most EUPs). (Segal 2007). Because of this, the set of problems they face and the processes they use are distinct from that of both Professional Programmers and EUPs. (Segal 2007)

The Scientific Method

The Scientific Method, at a basic level, is characterized by three steps: observation of a phenomena, proposing an explanation, and testing the explanation. (Carey 2011) However, to follow this process in a rigorous, efficient, repeatable, sustainable, open, and honest manner, other steps are necessary.

The principle of "Replication" requires that the tests be repeated, to ensure that results do not result from chance, or mistakes on the part of the researcher. This repetition may take part in the original experimental set up by the original researchers, or by other scientists who may re-analyze data from the original experiment to verify the conclusions drawn, or may replicate the experimental setup and produce their own data. Certain activities support replication: publishing full details of the methodology used so that the experiment may be replicated, publishing complete results so that the results of future similar can be experiments can be compared to them.

Best Practices for Scientific Programming

In this section we will describe several best practices for scientific programming we have identified in the literature, and explain why each is important to ensure scientific validity. Many of these best practices have practical advantages, which have been well studied from the perspective of software engineering, but we seek to justify each from the viewpoint of good science. Many of these justifications will be made using analogy to non-programming ways of conducting science. Many of these concern reproducibility, internal validity, or peer review.

A Distinct Design Phase

Hypotheses should drive the design of the scientific apparatus used, not the other way round. Because of this, software that serves as a research tool should be explicitly designed. (Baxter 2006). In scientific programming, the design phase often includes a consideration of the form of the input data, the required form of the output data, and the steps (both cleaning and transformational), that will be necessary intermediate steps. (Baxter 2006, Guo 2012) This phase may include multiple people: domain experts to specify what the system must do, those who will be doing the coding, and those who will use the system in its final form. (Carver 2007) The presence of a distinct design phase is a best practice that we will investigate during our interviews.

Documentation

Documenting methods and results is crucial to good science; it allows scientists to record their experimental setup, so that experiments could be repeated and their results reproduced. (Baxter 2006) Because of this, lab notebooks have long been used to

record these details. (Guo 2012) Research also shows that scientists have trouble tracking which settings, code versions, and datasets correspond to which results mentally, as well as what scientific intentions these changes represent. (Guo 2012) In light of this, scientists should document their code, as well as relationships between the data sets it runs on, and the results it produces. The presence of this documentation is a best practice we will investigate during our interviews.

Use Existing, Trustworthy Code

Following established, standard methodologies allows one to accomplish a certain scientific task without having to rigorously prove its validity. (National Academy of Sciences 1992) Using previously vetted external code is an analogous concept. Using existing scientific codebases allows shorter development times because less code must be written from scratch. (Carver 2007, Baxter 2006) However, using commercial code incurs the risk that a product may be discontinued, or that support may be dropped. Using open source solutions lessens this risk by allowing in house developers to assume responsibility for external code should community support be dropped. Additionally, using trusted codebases adds credibility to software quality, strengthening the internal validity of the experiment when external parties review parts of the code used in the experiment. (Baxter 2006) The use of existing, trustworthy code is a best practice we will investigate during our interviews.

Use Formal Version Control

Being able to demonstrate that results are reproducible is required to demonstrate scientific validity (Carver 2007). Additionally, it is helpful for scientists to know what dataset matched with which experimental setup corresponds to which set of

results. (Carver 2007) Scientists often recognize the need, even if they do not act on it, to track old versions of files so that they can repeat analyses or make comparisons between different outputs. (Guo 2012) Version control is used in both industrial and open source projects to achieve these aims. The extent and manner in which participants use version control is a best practice we will investigate during our interviews.

Testing

Testing scientific code is important to ensure the internal validity of the experiment that it supports. Internal validity refers to how well an experiment is able to establish that an observed result is caused by a known independent variable and not by some other phenomena (known as a confound). (Moring 2016) Within the context of scientific programming, inconsistent or buggy code poses a threat to internal validity (Baxter 2006), and this may be further problematic when the code's output cannot be validated against an existing system (Carver 2007). Testing, where each component of a program is checked for correct output, is one means of ensuring internal validity. (Adrion 1962) Rigorous testing is important to ensuring scientific validity. The extent and manner in which that participants test their code is a best practice we will investigate during our interviews.

Public Release

Peer review, where other researchers can critically examine the validity of methods and their results, is one of the hallmarks of modern science. Doing so requires that the work be reported unambiguously and anonymously, with full details of the design and methodology laid bare. (Hames 2007) In scientific programming, where code is at the heart of the methodology, it would seem logical that scientific code be

published. However, rarely is code published or even preserved, and many reasons are cited for this surprising phenomena: 1) the fact that it is neither accepted nor required practice, 2) packaging and polishing code takes time away from producing more research, and 3) fear that publishing code will impose a further responsibility for its creators to support others as they attempt to use the code. (Barnes 2010) However, these reasons do not negate the aforementioned scientific impetus for publishing code. The extent and manner in which participants publicly release their code is a best practice we will investigate during our interviews.

Case Study Methodology

Participants

Candidates for our study were recruited by asking existing contacts for "faculty and Ph.D. students in the sciences who may use code as part of their academic work." In departments where the we had no existing contacts but where we expected that code might be used in scientific work, we called or emailed departmental secretaries asking for candidates. Candidates were then emailed a brief description of the study and a request to participate. If the candidates agreed to participate, we scheduled a one-hour interview time slot.

A total of 11 participants were recruited in this manner. Care was taken to ensure a sample of participants were from as many different scientific fields as possible to aid the generalizability of any findings. Three participants were Assistant Professors, four were Full Professors, three participants were Ph.D. students, and one was a Postdoctoral Research Associate. Participants were polled as to what percent of their time they spent writing code at work, and the modal bracket was 20-30%, and the mean was 28%. In general, the more academically senior the participant was, the less likely they were to spend large portions of their working time writing code.

P#	Field	Job Title	Highest Degree Completed	% of work time writing code
1	Mathematics	Assistant Professor	PhD, Mathematics	20-30%
2	Astrophysics	Professor	PhD, Astrophysics	10-20%
3	Physics	Professor	PhD, Physics	10-20%
4	Chemistry	PhD Student	BS, Chemistry	40-60%

P#	Field	Job Title	Highest Degree Completed	% of work time writing code
5	Climatology	Professor	PhD, Geography	20-30%
6	Geographic Information Science	Assistant Professor	PhD, Geography	0-10%
7	Geography	PhD Student	MS, Computer Science	20-30%
8	Ecology	Postdoctoral Research Associate	PhD, Plant Biology	80-100%
9	Bioinformatics	Professor	PhD, Computer Science	0-10%
10	Biology	Professor	PhD, Biology	0-10%
11	Business	PhD Student	MBA	40-60%

Table 1: Participant Information

Materials

Participants were asked to fill out an online Pre-Interview Questionnaire in advance of their interview (see appendix). During their interview, participants were asked a standard set of seven interview prompts (see appendix). The Pre-Interview Questionnaire was designed to provide context to the interviewer so that more relevant follow-up questions could be asked. We collected information about the participant's job, prior training, purpose of the code they write, and limited information about the participant's coding process. Early responses from this questionnaire were used to refine the interview prompts, but the prompts were not modified after the first interview was conducted.

During the interview, participants were then asked a set of seven questions about the purpose of the code they write, the processes and tools they use to write it, challenges they face while writing it, how they learned to code, and how they thought the use of code in their field will change as time progresses.

Procedure

Participants were asked to fill out the “Pre-Interview Questionnaire” in advance of their time slot. If participants had still not filled out the questionnaire a day before their time slot, they were reminded. If they still did not fill it out, they were asked to do so during their time slot. This survey took participants an average of 10 minutes to fill out.

During their scheduled time slot, an overview of the study was explained to the participant, and any questions were answered. Participants were then given a Consent Form to review, and if they did not have any objections, sign. The audio recording was then started. The interviewer then asked questions found in the “Interview Script” (see appendix) to the participant, while writing the participant’s summarized answers into the form. Follow up questions deemed important by the interviewer were asked, and the answers recorded in an additional box set aside for them at the end of the form. After all questions had been asked, the audio recording was stopped, the participant was thanked, and any additional questions they may have thought of about the study were answered. These interviews took between 30 and 60 minutes to compete.

Scientific Programmer Case Studies

Participant 1

Thoughts on Learning to Code

Participant 1 has been learning to code since he was a child. He took programming classes in university, and coding came to be an integral part of his research as he pursued and earned a PhD in Mathematics. Nowadays he learns informally, by consulting online documentation when trying to learn how to use a language and by asking on Q/A site StackOverflow when he has specific questions he cannot find answers to elsewhere.

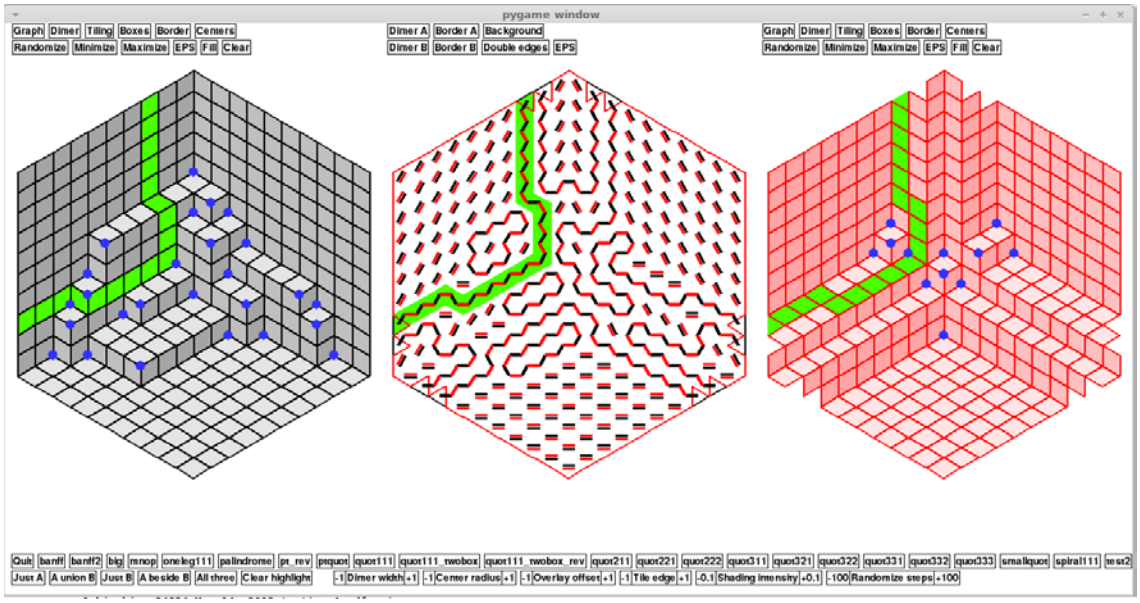
When he is helping new PhD students learn to code, he usually suggests that they try to do projects on project Euler, a mathematical programming challenge site. After this, he will give them a programming challenge more suited to their research area. In general, he recommends that new programmers try pair programming with a more experienced programmer, because as he sees it, this is the only way to learn Test Driven Development and Agile methods correctly. He also suggests that in general, people make a quick and dirty solution first and refactor it later.

Why they use code

Participant 1 is an experimental mathematician, and uses code to generate sequences, which are the object of his research. He also uses code to discover patterns in these generated sequences, and he uses these guesses as starting points for further manual analysis. As he explains, “the computer is the lab, and the computer allows him to have more data to look at”. He explains that without code, it would be far to time consuming to manually calculate the sequences he studies.

Additionally, he uses code to draw, as he puts it, “mathematical pictures”, which represent his mathematical discoveries. He says that these findings would be very difficult to fully explain without the use of these pictures, and further that these pictures would be very difficult to generate without the use of code.

Figure 1: An example of the “mathematical pictures” Participant 1 generates with code



What causes them difficulty

Participant 1 is a fairly experienced programmer, but expressed a few high level problems with his process. First, he noted that research projects that require him to write many lines of code usually end up being prioritized last. In this way, he does not like that the number of lines of code required to pursue a question determines the kinds of questions he is able to pursue.

Second, he is annoyed that he cannot embed his code in with his journal papers. He says that while “software is not a research project”, he does say “it is an experiment” and thus he feels he should be able to include at least key algorithms, but on multiple occasions when he has done so, the editor has asked him to remove his code from his

manuscript. He thinks that the way that research and researchers are evaluated should include a greater emphasis on code contributions.

Third, he finds it difficult to keep track of the structure of old code projects, especially because he does not document his code except for writing its associated journal article. Thus, he finds it difficult to remember how old projects worked.

What tools they use

Participant 1 uses Python, because he finds it expressive and fairly quick use to build things. He also says he has a lot of experience with its peculiarities and idioms. However in his view, Python has poor support for multithreading and application packaging. On the seldom occasion that he needs to profile Python applications, he uses GProf.

He wishes that the Sage mathematics library, which he uses extensively, were better indexed. He often has to determine if given functionality is already implemented in Sage, and he says it is often difficult to find this out. He finds that Sage can be difficult to debug, because it hides the stack trace when errors are thrown. However, he likes that it is open and developed by fellow mathematicians, which means that it usually fits his needs well.

He uses git for version control.

Where they want to improve

Participant 1 said that if he had more time to seek better training, he would probably learn to do Test Driven Development properly. Also, right now he writes code procedurally, and refactors to use an object oriented style. He says that with more training, he feels he would be able to use object oriented style “from the get go”.

Additionally, he feels that if he had experience as a Software Engineer, he would be more fluent with the git version control tool.

How they value code in their discipline

Participant 1 says that most researchers in his discipline who are in his age range are proficient in code, and that most people in his field use Sage, illustrating how important code is to his area of mathematics research. In addition, he says that he thinks that soon, there will be a coding course for mathematics PhD students so that they can learn to code in support of their research.

Participant 2

Thoughts on learning to code

Participant 2 took a few coding classes in college, but mainly learned to code on the job. He also said he has taken a few coding classes at his work. Coding classes were required as part of the physics curriculum during his undergraduate degree, and he believes this should also be the case at his current institution. He expects all of his PhD students to know how to code, but if they are just starting out, he recommends they use online tutorials, and he will also get them started on a really simple physics project where he will ask them to "reinvent the wheel", working on a coding problem with a known solution.

Why they use code

Participant 2 does research in astrophysics, and uses code to run hydrodynamics simulations, to perform data analysis, and as a teaching tool. He uses code to simulate and calculate properties of hydrodynamics configurations, such as analyzing the stability processes of rotating stars. He says that because he does non-linear calculations, it is easy to get them wrong, and thus easy to generate bad, spurious

results, and because of this, he uses code to validate his results using linear approximations.

What causes them difficulty

Participant 2 finds it difficult to debug his code, because he has to step through each line of code using a debugger and do a lot of math to manually verify each intermediate number. Before usable debuggers were available, he had to do this by using print statements. Once the code runs, it must be compared to other published results, which are based on experimentally derived physical realities. This process takes from a few months to a year. He wants to be able to parallelize his workflow, but he says that Matlab and Python prevent him from doing this easily.

What tools they use

Participant 2 uses C++ and Fortran. He also uses Matlab to set up the simulations, to set initial conditions, and to interface with the lower level languages, but is trying to switch to using Python instead because he says Matlab costs too much. He also uses GitHub to release code, and to do version control.

Where they want to improve

Participant 2 wants to become more efficient at writing code, because he often writes using trial and error. He also feels he often doesn't know about small "tweaks" might make his code run better or more efficiently, an example he gives is that he might be able to write code that finds the sum of a series of numbers, but might not be able to do it in the shortest or most computationally efficient way.

He want to become better about initial planning, and that rather than just having a general notion of what he wants to do he wants to have a better idea of intended functionality from the start.

How they value code in their discipline

Participant 2 says that being able to program is already a mandatory skill to do research in his area. However, he notes that because of the advent of higher languages such as Matlab and Python, you no longer have to know to do low level calculations, which in turn makes this mandatory skill more accessible.

Participant 3

Thoughts on learning to code

Participant 3 learned to code as a graduate student, by looking at code others had written, and by occasionally looking at textbooks on the languages he was using.

He says that conferences in his discipline occasionally have workshops to teach coding skills, usually lead by post-docs, and he would recommend that new PhD students use these opportunities to develop coding skills.

Why they use code

At a high level, Participant 3 uses code to analyze experimental sensor data. Before the advent of code, similar kinds of data analysis had to be done with by looking at high frequency photographs, and this technique only allowed approximately one frame every two seconds to be analyzed, whereas with code, researchers can analyze on the order tens of thousands of frames of data per second. Participant 3 said that aside from simply reducing the manual labor burden of his kind of research, this ability to analyze more data has enabled new scientific discoveries that are qualitatively different.

What causes them difficulty

Participant 3 said that using good coding style is difficult. Included in this is ensuring that code is easily reviewable and readable, and that he often has to go back and include better documentation through the use of code comments.

Participant 3 also said he has major problems with the increasing shift of the heavy lifting of computation to the “Cloud”. He feels the Cloud is too opaque, using the

words “black box” to describe it, explaining that as a classically trained Physicist, he likes to be “close to his instruments”.

What tools they use

Participant 3 works on a collaborative codebase, which has stringent rules that must be followed. One of these practices was that he contributes to a central wiki page which serves as documentation for the codebase, and also serves as the organizing structure to coordinate the coding effort of different team members.

He chooses to use TextEdit as his editor. He uses the Python “Matplotlib” library to produce charts and visualizations. Before Python he was using MATLAB, but he found that Python allowed him to interface with data easier and faster. Many of the tools he uses have backends written in C, with their interfaces written in Python.

Where they want to improve

Participant 3 feels that if he had more experience writing code, he would feel more confident pursing research that requires more complex coding projects, instead of simply making small changes to preexisting tools.

How they value code in their discipline

Participant 3 says that being able to code is already a skill that is required to do the kind of research he does. When searching for potential students to join his lab, he states that “What programming experience do you have?” is the first question he asks. When he was in graduate school, taking courses in software development was a requirement, but that such courses are not a part of the undergraduate physics curriculum per se, though they do have lab courses where students explore a physics topic through the use of code.

Participant 4

Thoughts on learning to code

Participant 4 originally learned to code using online tutorials, as well as more formal online courses. However, he learned a lot of his software engineering skills from a workshop put on by the Software Sustainability Institute, a consortium of UK universities dedicated to helping support researchers and software engineers. The workshop was specifically designed to teach software engineering skills to people in his field of research, with the view that by writing better software, the "peer reviewability" and reusability of research code. From this workshop, he says he learned things like how to package software for release, and how to have different modules written in different languages communicate effectively.

Were he to advise a new PhD student about how to learn to code, he would suggest that they learn languages that would strike a balance between usability and computational efficiency. He would also suggest taking a computer science course, because he thinks this may teach both a language and the proper conventions which may make it easier to write sustainable software.

Why they use code

Participant 4 writes code that analyzes output from established simulation packages. More specifically, he analyzes kinetic models of molecules using Markov state models. He says that this kind of modeling would be impossible without computer code, because it depends on performing calculations on many numbers, and also because to view the output, you must look through multiple graphs which would be impossible to produce manually. In particular, to simulate accurately, one must simulate at a small enough timescale that one can assume the molecules undergo constant acceleration, and for this assumption to hold, one must use a frequency of calculation that would prohibit manual calculation. To be clear, he did not design the simulation,

but his code sets up parameters for an established simulation package and analyzes the output data it produces.

What causes them difficulty

Participant 4 says that one of the biggest challenges he faces is designing how he wants the code to work, as well as creating an effective user interface. He wants others, both scientists in his lab and outside of it, to be able to use his code, but he finds that if he builds his code "too opaquely" people do not trust the "under the hood stuff". At a high level, his code unifies analysis at a large timescale with similar analysis conducted at a small timescale, and the design decisions he makes to unify these two analyses can lead people not to trust that his code operates correctly. Additionally, he must use Fortran for the heavy number crunching due to speed of execution constraints, but he uses Python for front end interfaces because he thinks it will allow others to make use of his code more easily, as well as allow him to display and visualize results more easily. However, bridging these two programming languages adds an additional layer of opacity contributing to others having what he describes as a low level of trust in his code.

While he is fairly confident in his coding ability right now, he remembers that while learning to code, he found it very technically difficult to link his Fortran backend to his Python interface.

What tools they use

Participant 4 uses Jupyter notebooks to structure a lot of his code. He uses different cells to build different modules, and likes that he can use this structure to break down code into chunks, that let him know where problems are. Additionally, he likes that he can serve Jupyter notebooks on a server containing his large datasets, and thus

access and modify his computations from remote locations, allowing him to do work and present his results wherever he chooses.

He uses the F2PY (which stands for “Fortran to Python”) interface generator to help bridge pieces of Fortran and Python code. He does not use an IDE, and uses Atom to write and edit code. He likes Atom because it is built into Github, so he can track which code has been committed easily, and also because it has a built in file browser organized in a tree.

He uses GitHub for version control and simply to store his code. He likes that GitHub lets you display Jupyter Notebooks on the GitHub website, allowing you to show working examples of your code in a web browser.

The hardware he uses is more powerful than average: he has a server with 32GB of RAM to handle the computation, and it runs Jupyter Hub, the server software which allows him to access Jupyter notebooks which can make use of to the hosted data from anywhere.

Where they want to improve

Participant 4 is confident in his process. He says that because he often interacts with computer scientists, he has been able to learn from them, including such lessons as code packaging and modular design. He also refers to the workshop he attended as making him care deeply about making his code as generally useful and usable as possible.

How they value code in their discipline

Participant 4 says that in his specific field, coding is already a required skill. Some people merely build scripts to connect existing code, but many are also able to build custom applications that fit their specific needs. He says there is a big trend of

learning to use NumPy and other Python packages among scientists as of late. He also gave a specific example of a colleague who had done this, and how this had let him shorten the time it took to analyze his data from a few days to just one day, which in turn allows him to use his time to solve problems more fundamental to his research.

Participant 5

Thoughts on learning to code

Participant 5 learned to code as part of his work, mainly by reading textbooks and by using Google to search when he found an error he did not know how to solve. He is a Professor, and when he teaches new graduate students to program, he "strong-arms" them into learning Fortran, usually by walking them through a simple program, and asking them to make simple modifications, and then by asking them to build something simple from scratch, such as to write a program that calculates the standard deviation of some numbers.

Why they use code

Participant 5 says he mainly uses code to make maps so that he can analyze spatial data visually. He works with climate models, and uses these maps to compare different models qualitatively. His code also reduces the data from the model to an appropriate timescale so it can be correctly compared with other data, which may be in a different timescale.

What causes them difficulty

Participant 5 says that the hardest part of his process is what he calls "code and data archeology". He says that it is hard to first find and then understand old pieces of code he has written. For data, he says he finds it hard to understand the purpose of input and output files long after the project with which they were associated has been

completed. He is often faced with the choice of writing code from scratch, or taking time to find code from old projects, which would serve the same purpose.

What tools they use

Participant 5 uses the Intel compiler in Visual Studio to build Fortran code. He says that Fortran is the fastest language to process his data, because it allows him to exploit parallelism in data reduction. He has notable hardware to process the large amount of data he uses: a computer with 32 cores, and a server with 32 cores and 192GB of RAM, as well as multiple 6TB hard drives. He uses Kedit and Sublime Text to write code. Kedit is a text editor that he uses for "looking at data and manipulating data". He also uses a dual pane file explorer called "Total Commander" to look through his code and data files.

Where they want to improve

If he had more time to learn, Participant 5 would work on learning formal version control, which he thinks would help him with his problems with "data and code archeology" discussed earlier.

How they value code in their discipline

Participant 5 says that coding ability is important in his discipline (Geography), so much so that he thinks there should be more departmental support to teach geographers to code. He thinks that undergraduates, and especially graduate students should learn Python specifically.

Participant 6

Thoughts on learning to code

In his role as a professor, Participant 6 regularly guides graduate students as they learn to program. He suggests that they take a Python class offered by his university's Computer Science department, as well as a "spatial analysis" course he teaches using

the programming language R. When he himself was learning to program, he took an intro to Java class during his masters degree, and also used Google and Stack Overflow to search online for answers to specific problems.

Why they use code

Participant 6 uses code to do simulation modeling, to do spatial analysis, to compute statistics, and to automate his workflow. He simulates forest ecology, and uses this simulation to study the spread of tree diseases. He says without code, it is impossible to do simulation modeling, because no existing software package would allow him to model the natural phenomena he studies. He says that it would be slightly more possible to do spatial analysis using a software package like Microsoft Excel, but that he often needs to use statistical methods which require more control than such software packages can give. Participant 6 (Assistant Professor) oversees Participant 8 (Post-Doctoral Research Associate), thus there is significant overlap in their two use cases because they work on overlapping projects.

What causes them difficulty

Participant 6 feels that dealing with abstraction is his biggest difficulty. This is a classic problem when modeling: he wants to ensure that he can build an abstract model of natural phenomena, but without loosing important detail required for the model to be usefully accurate. He says that when he builds models that take into account “too many parameters”, he has to worry about these models being computationally efficient.

Related to this, he says it is difficult to get his collaborators to agree on model parameters, both in terms of which conceptually relevant at all, and on which are important enough to be included in the model. This problem is compounded by the fact that he works with a diverse team spanning many disciplines (including political

science, climatology, ecology, wild fire science, and computer science) and that not all of them understand the difficulty of building a simulation model in code. They also often disagree on the degree of granularity that parameters should be included (e.g., modeling on a spatial resolution of hectares vs acres).

What tools they use

He uses R to compute statistics and to do spatial analysis. He likes it because it has what he calls “the best support community”, as well as many libraries that make it easier for him to process spatial data.

In addition to using simulation models in his research, he also teaches students how to use them in class. For this, he uses an agent-based programming language and integrated modeling environment named NetLogo. He like this because he feels it is easy to learn and teach, but is still reasonably powerful.

His Post-Doctoral Research Associate is largely responsible for building the simulation model, so a discussion of the tools used to build this will be discussed in the discussion of Participant 8.

Where they want to improve

Participant 6 wishes he had more time to learn to program better so that he could be more in charge of developing the models, which he currently assigns to his postdoc. He says that if he had more coding ability, he would have done some of the coding for these projects, and also would have been able to contribute more to the technical design discussions that go into creating a model. He also says that this way, he would have been able to build models faster to demonstrate them to other, less technically inclined team members.

If he had better technical skills, he also says that he would have been able to put more emphasis on usability, particularly so that other people could open up the model and change the parameters more easily, and furthermore, allow him to package his model in a way that would be of more general use.

How they value code in their discipline

Participant 6 says that being able to code, at least in some capacity, is already a required skill in his discipline, and that this requirement will only become more stringent in future. He says that in future, it will become expected that PhD students know how to code, especially if they want to get a job in an industry where they work with data after they graduate.

Participant 7

Thoughts on learning to code

Participant 7 has a Bachelors and Masters Degree in Computer Science, but now is pursuing a PhD in Geography, and of our participants, thus had the second most formal training in Computer Science. As expected, she says that it is through both of these degrees that she has obtained most of her coding ability.

When asked how she would help a new research contributor learn to code, she said that she would suggest joining a project lead by a more experienced coder, and start with small coding tasks to learn in a "hands on" manner. She is careful to emphasize that she finds it more important to "learn how to complete tasks" rather than spend time "learning the basics of a language". She also lists "[learning] versioning, excessive commenting, testing often" as the top three most important things coders should learn to do. After doing this, Participant 7 suggests taking an "Intro To Python Programming" course in the computer science department.

Why they use code

Participant 7 uses code to allow people with blindness to access the information, which would normally be experienced visually through a geographic map. To do this, she builds a dynamic vocal auditory interface to allow them to experience spatial data. She also uses eye tracking to understand what data is most useful to sighted people, and needs code to collect this data. She also uses code to process the data she collects. Without code, she supposes that she could record the spatial information she wants to convey, but says that her system would lack the interactive element she wants it to have.

What causes them difficulty

Participant 7 expresses three difficulties. First, she says that she has often had to change her process to accommodate people with less coding experience, often by omitting the use of tools she finds supremely helpful. One example she gives is that she has had to stop using formal version control because her collaborators did not know how to use it, and it was deemed easier to stop using it than teach the new collaborators how to use it. She said that she has had more positive experiences too, where she has had what she termed “workshop days” where non technical collaborators would all work in the same room as the CS-trained collaborators, where they were able to make faster incremental progress on their project using paper and whiteboard and back and forth direct interaction.

Another thing she found difficult was that a library (GeoTools) she uses was designed to process spatial information, using the “Java Swing” graphical user interface toolkit, but that she found it difficult to adapt the GeoTools library to a non-visual approach to outputting spatial data.

Finally, she finds that when working in a group of diverse levels of technical expertise, she finds it is hard to “articulate structures and ways of thinking about [technical] things which would make it easier to reason through a [code] process”. She characterizes this as a knowledge gap which makes communication more difficult, and one example she gives is trying to explain what a design pattern is.

What tools they use

Participant 7 is fairly happy with her tools, but as mentioned previously, she dislikes that she occasionally has to drop the use of certain tools to accommodate less technically inclined collaborators.

She uses Java, as well as the Java Sound API to build her spatial data auditory interface. She uses GeoTools, a Java based Geographic Information Systems ToolKit, which allows her to process spatial data. She uses Apache Subversion for version control, and the Eclipse IDE to write her code. When she needs a text editor, she uses TextMate.

Where they want to improve

Participant 7 mentions that in a research setting, quality is often “good enough” to meet the fast paced needs of research, but given more time, she would write “cleaner” code and would document her code more thoroughly by including more comments.

How they value code in their discipline

Participant 7 says that her field of research, Geographic Information Systems, requires that people be able to code. She says this need is less present in Physical and Human geography.

Participant 8

Thoughts on learning to code

Participant 8 learned to code using online courses, such as those for Java and C++ available on Coursera, as well as through textbooks such as “Head First Java”. He also mentioned that he found YouTube tutorials and StackOverflow questions and answers to be particularly helpful. If he had the opportunity, he says he would have taken an Introduction to Programming course offered in the Computer Science department at this university, where he imagines he would have learned both language skills as well as good coding practices which he characterizes as harder to pick up by oneself.

Were he to give advice to someone learning to code for research purposes, he would suggest they start working on very small pieces of a project, erstwhile working through an accessible book like “Head First Java”. He would also suggest they take the time to learn how to use debugging tools early on, because he feels this will save a lot of time in the long run. He emphasizes the importance of commenting code, reflecting that he has spent hours looking at old code to try and understand what it does.

Why they use code

Participant 8 works as a Post Doc under Participant 6 to build an agent based model of invasive grasses and other floral phenomena. He explains that in effect, this means he “builds a world in a computer, gives agents properties and behaviors so that they can interact with other agents”. He uses his own and other’s knowledge of invasive grasses to translate natural phenomena into a model that he can build in code. He says that the purpose of a model is less to predict the future with precision, but more to bracket the range of possible outcomes given hypothetical parameters.

Participant 8 also uses code to do statistical analysis on large datasets that would choke software packages such as Microsoft Excel. He also uses code to exert a high degree of control when making charts and figures for publication.

What causes them difficulty

Participant 8 says that he finds troubleshooting and debugging to be one of his hardest tasks. He says this is largely because he is typically learning to use the language he's using while simultaneously using it to build complex models, and his unfamiliarity with the language and the algorithms he's using makes them hard to debug. He regularly encounters both fundamental flaws in his program, as well as small "one off" errors.

He also has difficulty communicating the work he's done with those who are less familiar with coding or those who are less familiar with the project, and packaging up his code so that it's usable by these people. He says because of this technical knowledge gap, it is hard to elicit feedback on his model from those with domain knowledge.

He says that it is hard to learn new classes of algorithms. One example he gives is that he had to learn how to do linear programming for an optimization problem, but didn't know any linear algebra, so he wasn't even aware that linear programming could do for him. He says that as a result, he had to learn a new technique as well as its specific implementation.

What tools they use

Participant 8 uses R for analysis and for producing graphics. He has thought of using Python, but he is comfortable and has so much experience using R that he does

not feel compelled to switch. He says that R is not good for modeling, because there is a lot of overhead, and because it is slow.

He uses Java to build his agent-based model using a modeling framework named Repast, which includes visualization tools. He uses Eclipse for his IDE, and likes that it has different perspectives for programming and for debugging. He also uses JUnitTest to test his code.

Finally, he has taught courses in NetLogo, a lightweight agent-based modeling system, but does not use it for his own research though others do, because he finds that it gets really slow when you use it to build large models.

Where they want to improve

While Participant 8 is fairly comfortable with his process, he says that he wishes he had a more formal way of visualizing projects from the get go, including how code would have been written and the stages it would go through. He also wishes he had more time to write formal test batteries.

How they value code in their discipline

Participant 8 says that code is already a required skill in his area of research, and certainly for those who wish to build and tweak agent based models. He says that even in Ecology and in the Social Sciences, you need to use R or something like it for spatial analysis, because GUI software packages are limited and often do not include the functionality required.

Participant 9

Thoughts on learning to code

Participant 9 is in a fairly unique situation with respect to his CS experience. Though his current appointment is in Biology he was formerly appointed in the Computer Science Department, and he has a PhD in Computer Science and does

research in Bioinformatics. This means he has a large amount of coding and theoretical experience, as well as a great degree of formal training in the field.

If he were to help a new PhD student learn to code, he would suggest that they write code into Jupyter notebooks, a platform which allows people to embed rich text and graphic documentation with executable code cells. Occasionally, he refers them to a an introductory programming textbook. He also says that there is a current graduate level experimental course offered named “Programming for Biologists”, which aims to give biologists just enough programming so that they can automate and much of their results analyses. He also says that he likes the website “SoftwareCarpentry.org", a website specifically designed to teach programming skills to scientists. They also often offer 1-2 day intensive tutorials at scientific conferences and such.

He is careful to make the point that he thinks biologists do not need to become well trained programmers, but that often they just need enough skills to do their work and not much else.

Why they use code

Participant 9 uses code for lower level tasks he terms “workflow management”, but because he is in Bioinformatics, code is often the language of his research.

Participant 9 described workflow management as connecting existing tools which must be used sequentially to find results. He writes scripts that connect these tools to allow him to work faster and process more data.

However, Participant 9’s main use of programming is far more novel. Using his background in Computer Science, he takes an information theory approach to gene sequence alignment, which is a common problem in biology. At a high level, he explains that classically, gene sequence alignments were computed and represented by

the “longest common subsequence”, that is the longest string of common elements between two genes. However, he explains that this is not a useful way to represent similarity, because it describes similarity effectively by describing gaps in similarity, rather than describing overall similarity directly. Instead, he describes his approach to describing similarity as “the amount of information required to describe an unknown sequence given a known sequence”, where two highly similar sequences will require less information and two highly different sequences will require more information. He says that he wanted to use this new approach because he thinks it is a more natural and unbiased way to represent similarity, alleging that the old way had so many parameters such that it could be manipulated to render a desired result, and is thus is more subjective.

What causes them difficulty

Participant 9 says he often wishes that he spent more time building a test suite and automating unit tests. He thinks this would allow more efficient unit tests throughout the code writing process, and allow him to spot problems earlier on.

He says that he often finds algorithms conceptually difficult to reason about and build, but he describes this as “the fun part”.

He also says that he dislikes doing “system administration type stuff”, such as when package updates cause his code to break. He says that he often has to go back and fix everything after an update, but this is “becoming more sane” now. Similarly, he also finds makefiles and configuration files hard to grapple with.

What tools they use

Participant 9 uses C++ for his novel sequence alignment research. He uses Textmate as his editor, which integrates a tree style file explorer. He uses a document

management program called DEVONThink Pro to keep a virtual lab notebook. Participant 9 uses Jupyter notebooks to teach computer science courses. He partially fills in code cells, and asks them to complete desired functionality as an assignment. He says he sometimes also uses Jupyter notebooks for research, because he likes that he can integrate his documentation with his code.

Where they want to improve

Participant 9 has a PhD in Computer Science and thus much computer science experience, and he says that because of this, he is fairly happy with his process.

How they value code in their discipline

Participant 9 has a unique perspective, having formerly been appointed as a Professor of Computer Science but with a current appointment as a Professor of Biology. In his current appointment, he heads a curriculum reform committee charged to modify the undergrad curriculum to add more quantitative skills such as mathematical modeling, statistics, and computer programming. He says he reflects his opinion that biologists need more quantitative skills, because the nature of biological data is changing: with cheap gene sequences, more biological discoveries are possible but only through the use of code to process this large amount of data. He also believes that code contributions are more valued in the field of Biology than in Computer Science.

Participant 10

Thoughts on learning to code

Participant 10 has been taking programming classes since high school, reminiscing that his high school was given one of the first mass production minicomputers on which he learned to write BASIC. Later, he was introduced to coding in a research setting by a post doctoral research fellow.

Where he to coach a new PhD student as they learned to code, he would sit next to them and instruct them while they code and introduce them to basic concepts such as variable declarations, if statements, and for loops. Then, he would give them a small biology related problem to solve.

Why they use code

Participant 10 uses code to model worm feeding behavior. His code processes video data of worms crawling on agar plates searching for food, and analyzes this data to find hidden Markov models, to model this behavior. He says that without code, there would be no way to do this work by hand, because there would be no way to analyze this behavior on the thousands of experiments their models are based on, and doing it with fewer experiments would not allow them to find statistically significant results. His code also does math that is very difficult to do manually.

What causes them difficulty

Participant 10 says that it was very hard to interpret and analyze video data, because there were many artifacts in the video, such as vibrations, for which he had to compensate.

He also wants to put more effort into improving his tool's user interface, but also does not want to spend too much time on this task because there are more pressing concerns, and because his tool is not designed for general use. He finds this tradeoff difficult to make.

What tools they use

Participant 10 uses a C-like compiled language called "Igor Pro", designed for interacting with experimental scientific data, which includes ample image processing operations. He says he likes it because although it is a compiled language, it compiles so quickly that it can be readily interacted with. He dislikes that "the built-in editor is

bad", because it requires that he use the mouse in addition to the keyboard, which he finds slow. He also says that though Igor is C-inspired, it has many idiosyncratic ways that it differs from C that can occasionally make it difficult to teach others how to use it.

Where they want to improve

He wishes that he was more experienced with a more conventional language or programming environment, because he feels that he would be able to build a more robust and easier to share product that way.

How they value code in their discipline

He says that in his lab at least, coding skills are already required. He also says that he is trying to bring coding instruction into the undergraduate Biology major. He feels this could be integrated as an active learning exercise, using biology inspired exercises.

Participant 11

Thoughts on learning to code

Participant 11 mainly used online resources to learn to code, relying heavily on Mathematica's online help. He has also taken live online classes, with instructors who answer your questions. This service was provided free to him through his University's institutional subscription. He says he has also taken many classes in Economics, and he feels that solving the assignments using Mathematica is a good way to use Mathematica.

Why they use code

Participant 11 uses code to do the mathematical heavy lifting to support his research in Business Operations. He does mathematical analysis of models. He finds optimal values for different "players" in a given model. He says that he could do

perhaps 45% of this math by hand, but the hardest part is transitioning between different states of his model, which he describes as “messy without code”

What causes them difficulty

He says that when building large complicated scripts in Mathematica, he often has to simplify things to allow him to verify that the script’s output is correct. However, he says that because of this simplification, he finds that he looses the ability to see the intuition behind the code.

He also finds it difficult to collaborate with coauthors when using code. This is because when everyone writes different parts of the Mathematica code required for a paper, it can be very difficult to understand each other’s code.

What tools they use

Participant 11 primarily uses Mathematica and Matlab. He finds that both packages have complementary strengths and weaknesses. He likes Mathematica because it can intelligently simplify mathematical functions. He also feels that Mathematica is good for visualization tasks, such as plotting functions. He uses Matlab less, but finds it is a better choice for numerical analysis.

Where they want to improve

He says he would like to learn a “normal” programming language like Python or R, because his colleagues in the Computer Science department have said that these languages would allow him to solve the same problems but in a more computationally efficient manner. He also dislikes that Mathematica is not very good at working on large datasets, and that more conventional programming languages may make this easier.

How they value code in their discipline

Participant 11 says that the use of code is emerging as an important tool in his area of research, because increasingly real business datasets are available to work on. He feels coding skills are not yet required, but that that in five to ten years, they will be.

Best Practices Evaluation

P #	Distinct Design Phase	Documentation	Existing, Trustworthy Code	Formal Version Control	Testing	Public Release
1	frame research conceptually, discuss requirements with collaborators	none beyond writing associated research paper	uses Sage libraries when possible	Git	tries but often fails to conform to Test Driven Development	Yes, on GitHub. Does not clean up.
2	looks at similar code, writes mathematical equations underlying code, builds flowchart, builds modularly from flowchart	did not discuss	based on legacy code from 1980s	Git, GitHub. does version control "very loosely, informally"	creates test problems based on physical phenomena, tests each module in pipeline individually	Yes, on GitHub
3	for big systems: draw diagrams on paper, assemble code fragments, ask about existing code from collaborators	peer code review to ensure code is "reviewable and readable", wiki page for higher level documentation	own experiments use code from codebase with 1000s of collaborators	Git	tests for known outcomes	does not publicly release
4	napkin drawing with collaborators, lit review, detailed written diagram, method and class definitions	documents using Jupyter Notebooks	Python APIs such as NumPy, Pandas, MatPlotLib	Git, GitHub	writes test cases with known values	Yes, on GitHub
5	talks with collaborators, search for suitable existing code, looks at input data and imagines desired output data	does not document code well enough, has to do "code archeology"	minimally, examples from textbook when using R	records version number and date for each code file	inspects output for "reasonable results"	Shares with collaborators, does not post publicly
6	designed as part of grant proposal, interdisciplinary discussion for conceptual design, pseudocode	did not discuss	does not reuse code	Git, GitHub	Sensitivity Analysis on his model	does not publicly release

P #	Distinct Design Phase	Documentation	Existing, Trustworthy Code	Formal Version Control	Testing	Public Release
7	look for existing libraries which may fulfill conceptual goals, additional reading, interactive design with lab meetings and prototype testing	feels less important to "clean up and comment" in research code as opposed to industry	reuses scripts, but doesn't reuse code for main application	stores in Apache Subversion repository	paper logic check, inspecting verbose output	Yes, via ScholarsBank and lab's website
8	reads through technical literature to translate conceptual design into technical design	finds scientific modeling code poorly documented, encourages learners to comment well	adapted from previous code, but current iteration bears no resemblance	Git, GitHub	unit tests with JUnit	Yes, as supplemental materials to papers, and direct to peers
9	minimal design process, created prototype then flesh out different parts	keeps lab journal, occasionally uses Jupyter Notebooks	sometimes reuses code from GitHub or academic's websites	Git	"add hoc" test suites	Yes, on GitHub and on lab's website
10	consider physical phenomena to model, work out mathematical underpinning, write flowchart outline with collaborators	did not discuss	does not reuse code	does not use version control	test with fake data with known answers, including extreme data	Yes, when journals accept code
11	lit review for similar code problems but rarely able to find code, choose mathematical functions	did not discuss	does not reuse code	stores different versions of code in different folders	verify mathematical results by hand	does not publicly release

Table 2: Summary of Evaluations

A Distinct Design Phase

Participant 1 said that his design process usually comprises framing a research question at the conceptual level and reading papers and discussing the code requirements with collaborators. He says he then writes inefficient but easily understood code first, before optimizing this code after he is happy with his design. **Participant 2** said that his design process begins by looking in the scientific literature and by talking to colleagues who may have written code which solve similar problems. He then writes down the mathematical equations underlying the eventual code on paper, and then he converts these equations into a flowchart representing the eventual modules of his system. He says his fundamental algorithms are often adapted from code has found in his literature search. He then writes each module of code, where each module of code corresponds to a distinct step in his scientific analysis. **Participant 3** says his design process often begins by considering the physical phenomena that serve as his data, and by searching for existing tools, which may be similar to those he is trying to build to interact with this data. When designing bigger systems, he will often draw diagrams on paper, and then assemble fragments of code he thinks will be useful, often asking collaborators if they know of any existing code which fulfills his requirements. **Participant 4** latest project's design phase started with a napkin drawing by one of his more senior collaborators, which lead to searching through scientific literature and a continuing series of conversations to discuss the technical requirements in greater depth. Before he began writing code, he made a diagram with a pen and paper, and then wrote the structure of code including method and class definitions. **Participant 5** begins his

design process by talking with collaborators, and by seeing if any code he has written before or may find in a public repository may fulfill a similar purpose to that which he is considering writing. He also looks at the data he has, and considers the form in which he wants the data to be in after being processed. **Participant 6** usually applies for funding for a research project which he knows will require the use of code, so he has thus already given quite a bit of thought to the design of his code before he begins building the system. His research often requires collaboration from an interdisciplinary team, and this team will often design the model to be built in code around a whiteboard over about two days. After collaborating with his less technically inclined colleagues, he then translates the model they agree on into pseudocode, and then his post doc (Participant 8) who is even more technically inclined will translate this pseudocode into real code. **Participant 7** usually begins by looking at what libraries may exist which align with her conceptual requirements. Based on what she finds, she does additional reading to understand similar solutions to the one she has in mind. She says that this often occurs in an iterative process of design, between lab meetings and discussion and prototype testing with volunteers. **Participant 8** works as a post doctoral research assistant recruited for his programming skills to work under Participant 6. Though he is not the one who is responsible for determining the original design, he often reads through a lot of literature to determine the technical design details after being passed the conceptual design by Participant 6. He will often read through graph theory literature to understand how the design of his model may borrow concepts from graph theory. **Participant 9**'s design process is minimal, because his code project began to investigate the possibility of implementing an information theory in a biological

context. He says that he did not design his system first, but instead began by creating a prototype and recruiting students to flesh out parts of the system. **Participant 10** designed his system by first considering the physical phenomena he wanted to model in code. Once he had worked out the mathematics underpinning his model, he then designed his code by writing a flowchart outline on a piece of paper in collaboration with a colleague working on the same project. **Participant 11** begins his design process with a literature review to see what similar problems have been solved using code, but is frustrated because he rarely is able to find the actual code used. After this, he chooses the mathematical functions he will have to implement, and then after this abbreviated design process, he begins building his system.

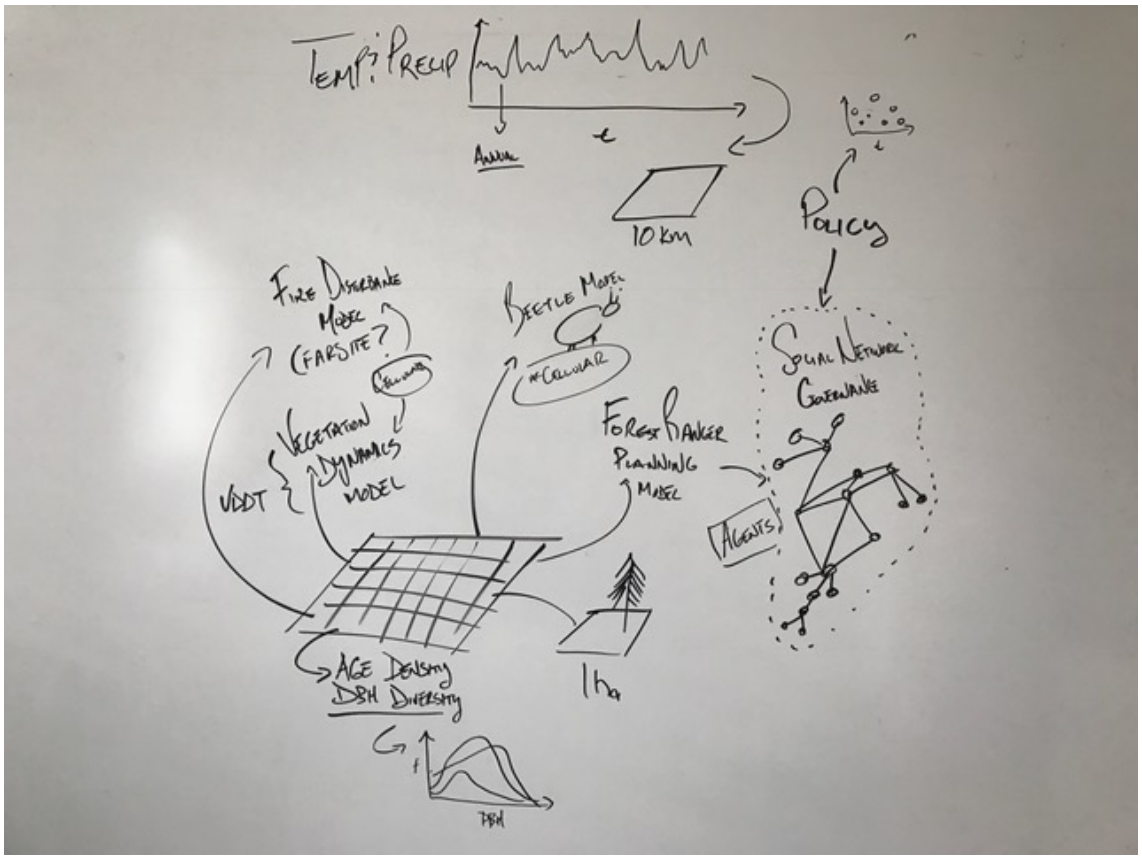


Figure 2: An example of Participant 6's conceptual models drawn on his whiteboard, forming part of his design process.

Due to the nature of academic research, there is often a natural and deliberate yet brief and informal design phase before coding starts: usually code requirements are often decided in response to a research question, and code requirements and design often arise organically out of the literature review and methodology design research processes. Thus we witness that the scientific programmers we studied usually have some sort of design process, but these processes are varied and often informal. All of our participants recognized the importance of having a design process, even if this resulted in wildly different levels of formality. Their design processes often included collaborators in their field, and also often included those with less coding experience than themselves.

Documentation

Participant 1 talked at length about why he does not document his code rigorously. In his view, the only documentation his job (Professor) incentivizes him to produce is the research paper his code supports, which only documents the results and purpose of his code at a very high level. He thus is incentivized to move on to producing more research rather than taking the time to document his code. He also mentions that if he tries to include snippets in his submitted papers, an editor will invariably ask that he remove it. As a result, he finds it difficult to understand old code he has written. This tension between the pressure to move on to publishing new research and the benefits of rigorous documentation is a theme expressed by many other participants. **Participant 3** explained that one of his biggest challenges was ensuring that his code is "easily reviewable and readable" by others. He says that he believes that collaborators reviewing each other's code is an important vetting process necessary to doing good science. He says that sometimes code is not transparent or its function is not obvious and thus people have to go back and improve their documentation. He often will use a wiki page for high level documentation. **Participant 4** mentioned that one of the reasons he uses Jupyter Notebooks is because it allows him to write code with interspersed rich text, allowing him to easily document his code. **Participant 5** mentions that he often writes code from scratch even though he knows he has written similar code before but does not want to do what he calls "code archeology" required to find and understand it. He also mentions that he is excited by a new tool called "R Studio", an online interface for the R programming language, which allows embedding of documentation and visualization in R code. However, he does not yet use this new

platform. **Participant 7** discussed how code quality seems to be less important of a concern when producing research software. Comparing her experience writing research software to her past experience writing code in industry, she stated that "research software is 'good enough', once it works you don't go back to clean it up or comment it". **Participant 8** mentioned how he has found it difficult to reuse scientific modeling code because he found it poorly documented and commented. He advises that people who are learning to program use good commenting style, should they ever need to understand their old code. **Participant 9** keeps a "lab journal" of sorts by describing his progress on his coding project in a plain text file as he progresses. He also sometimes uses Jupyter Notebooks in his research because he likes that he can easily integrate documentation with his code. **Participants 2, 6, 10 and 11** did not discuss their documentation processes when talking about their development process.

Attitudes towards and practices of producing documentation to accompany the code researchers write were mixed. Many participants produced documentation as part of their design process as discussed previously, but few preserved this documentation for the lifetime of their project or documented other parts of their development process. Four participants did not discuss documentation while talking about their development processes. Many scientific programmers do not recognize the importance of documenting their code, or recognized the importance of doing so but explained that external factors made it less of a priority. Two participants even talked at length about the problems neglecting to document their code directly causes them, yet still they do choose not to document their code.

Use Existing, Trustworthy Code

Participant 1 says that he always tries to find a Sage library which will perform the mathematical functionality he wants to implement, and will only write code from scratch if his search is unsuccessful. He also wishes that Sage libraries were better indexed, because he often finds it difficult to determine if his desired functionality already exists in a sage library. **Participant 2**'s codebase is based on hydrodynamics simulation code written in the 1980s, which his lab has taken and significantly modified. He also describes how he reuses code in a less strict sense, for example by talking to colleagues about how they may have solved a coding problem on a conceptual level, and then implementing their conceptual solutions in code. **Participant 3** contributes to a codebase with many (thousands) of distributed collaborators, and the code he uses in his own experiments is often pulled from this codebase. **Participant 4** uses many Python APIs in his code such as NumPy, Pandas and MatPlotLib. **Participant 5**'s level of code reuse depends on the language he is using. He says that he rarely reuses code when using Fortran but occasionally uses examples from textbooks when using R, and when using a visualization language named NCR, he often has to reuse code because he finds it too "clunky and difficult" to compose from scratch. **Participant 7** that if she is automating a well established data processing pipeline, she will reuse scripts from other research labs, but if she is writing software, she will start from scratch but use libraries as appropriate. **Participant 8**'s current project was adapted from code written from a previous Post Doctoral Researcher in his role, but he states that his current codebase bears little resemblance to the code he inherited.

Participant 9 only sometimes reuses code, finding on fellow academic's websites or downloading it from GitHub. **Participants 6, 10 and 11** say that they do not reuse code.

Seven of our eleven participants reuse code in some form, but this reuse runs the gamut from occasionally using scripts downloaded from other academics' websites, to making extensive use of libraries for the core functionality of their codebases. Participants often talked about considering code reuse as early as the design stage, as discussed above, where they would look for code that might help implement their conceptual goals. However, the majority of participants who reused code did so superficially. Two participants used libraries, and two participants' systems are adaptions from legacy code, which bear little resemblance to the original.

Use Formal Version Control

Participants 1, 2, 3, 4, 6, 8 and 9 use Git for at least one of their projects, and of those, **2, 4, 6 and 8** host their code on GitHub. **Participant 2** says that he usually does version control "very loosely, informally" for the bulk of his code, but uses GitHub for one of his projects but he finds it hard to convince everyone to use it. **Participant 5** says he simply records the version number and the date for each code file he writes, but does not use a more formal method of version control because most of what he writes is in his words "write once, use once". **Participant 7** stores her code in a repository, typically using the Apache Subversion tool. **Participant 10** does not use any version control for his project, which has been in development for 15 years. He believes that version control systems add too much overhead. **Participant 11** does version control very informally, storing different versions of his code in different folders.

Participants often said that they used a version control system, but later when talking about their process, they did not talk about how this version control system integrated with their process, suggesting low degree of commitment to using their chosen version control system. This was confirmed when a few participants enumerated perceived flaws of version control systems, such as the introduction of unnecessary overhead and the difficulty of achieving "buy in" from their whole project's team. Most participants are aware of version control systems and have used Git in some context. Four of these participants also say they release their code publicly on GitHub (discussed below), suggesting a higher degree of commitment to version control. At the same time, these participants did not articulate how they use these version control systems when talking about how they develop software.

Testing

Participant 1 says that he tries to adhere to Test Driven Development practices, but usually fails. Instead, he usually does unit tests after writing his code. Also, because he usually uses his code to compute sequences, he will compute the first few elements in the sequence by hand, and will compare them those his code generates. **Participant 2** says that he will create test problems based on known measurements of the physical phenomena he studies. He also has different modules strung together in a pipeline to output results, and he notes that he tests each one separately, doing the math by hand. **Participant 3** tests his code similarly, testing for known outcomes. **Participant 4** says he writes specific test cases with known values. **Participant 5** merely inspects his output to see if he finds them to be "reasonable results". **Participant 6**, who uses code to model natural phenomena, uses a technique called "sensitivity analysis" to test his

system. This technique is used to determine the how much of the uncertainty in the output of his model can be apportioned to the uncertainty of each provided input. **Participant 7** uses a "paper logic check" where she traces the logic of her code on paper, and also will make her program provide verbose output while running different pieces of her code to confirm each is working. **Participant 8** unit tests his code using JUnit. **Participant 9** builds ad-hoc test suites to test his code. **Participant 10** will generate fake data with known answers, and make sure that his code will replicate these known answers. He makes sure to include extreme (but correct) results in his tests. **Participant 11** will verify the mathematical results of the formulae his code implements manually, step by step.

Seven of our eleven participants said they tested their systems with some sort of data with known expected outputs, with varying degrees of formality and rigor. One used a dedicated testing tool (JUnit), but the rest tested their code with test data manually. Two of our eleven participants tested their system by merely inspecting its output. Two others tested their system by using a method suited specifically suited to their use case (sensitivity analysis, mathematical verification).

Public Release

Participants 1, 2, 4 and 9 all release their code publicly on GitHub. **Participant 1** notes that he does not attempt to package it nicely before doing so, and also notes that he wishes the journals he submits to were more open to accepting code as part of his submissions (discussed earlier). **Participant 9** will also release his code on his lab's website. **Participant 5** releases his Fortran and NCL code informally to collaborators and those who ask via email and FTP, but does not post it in a public

location. He hosts his R code on RMarkdown HTML websites. **Participant 7** releases her code via ScholarsBank and via her lab's website. **Participant 8** has released his code as supplemental material in publications, and he has also shared his code directly with peers. **Participant 10** releases his code via journals which accept and publish his code separately, but notes that not all journals do this. **Participants 3, 6, and 11** do not release their code publicly.

Seven of our eleven participants release their code publicly at least in some cases, and one of the remaining four shares his code when asked but does not release it publicly. Two of these seven participants submit code as part of their journal or conference submissions, and one additional participants wishes he could do so. We see that there is a high degree of willingness to release code among our participants.

Conclusions

On the whole, a low degree of adherence to best practices was observed. Nonetheless, a few noteworthy cases of the creative implementation of best practices leave room for optimism, and such cases be used as examples of what processes are likely to work in scientific programming contexts. Additionally, in interviews, participants often discussed problems which could be solved by adhering to best practices, and even directly and independently expressed the will to learn how to conform to the best practices we studied, suggesting that scientific programmers may be receptive to education about and the codification of such best practices.

Of the best practices we studied, the existence of a formal Design stage was the one we observed to be most closely adhered to. All participants recognized the importance of an explicit design stage, but not everyone produced tangible documents in response to this. We noticed that larger teams were more likely to produce written design documents, and that most design activity occurred in a collaborative meeting format involving all stakeholders in the research process. Therefore, we suggest that taking minutes at these meetings may be a first step to formalizing more intuitive, ad hoc design processes.

Overall, the importance and scope of Documentation was poorly understood. Many participants spoke of how they advise new coders to learn to comment their code, and many also complained about problems ostensibly resulting from poor documentation. One participant explicitly analyzed the reason he gives so little attention to documenting his code. Only two participants documented their process with

significant rigor, and both related their rigorous documentation process to their need to communicate with the rest of their development team.

The use of Existing, Trustworthy Code was rare. Only two participants made use of Libraries at all, four participants did not reuse code at all, and the rest mentioned code reuse that was insignificant. Of the two participants who did use libraries, both used languages known for an abundance of libraries, perhaps suggesting that the use of more modern and mainstream languages may encourage more to reuse code. However, not all participants who used such languages reused code, suggesting that increased education about the benefits and method of reusing code may help increase adherence to this best practice.

Many participants had heard of Formal Version Control systems, but few talked about them as if they were an integral part of their process, and many also expressed problems which could be solved by using version control systems, suggesting a low degree of commitment to their use. Increased education as well as more flexibility on the part of version control tools, will likely help scientists adhere to this best practice.

Most participants engaged in a reasonably methodical Testing procedure involving manually writing and running test cases, except for two participants for whom this method would not make sense, and two participants who had a much less methodical test procedure. Because writing and repeatedly running test cases is time consuming and can easily be automated using existing tools, increased education about such tools would likely lead to time savings and increased rigor.

Participants displayed a surprising willingness to Publicly Release their code. Eight participants publicly released their code in some form, and of those, three did so in a formal manner accompanying published papers.

By and large, it is our opinion that increased adherence to best practices could be achieved by a three-pronged approach: education, adaption and incentive. Many participants had problems, which existing tools and processes could solve with no modification, but they did not know of these tools or did not know how to use them. Increased **education**, both about the range of tools and processes available and about how to use specific tools and processes, can help solve these problems. A few participants said that they know of tools which almost but do not quite meet their needs, often because they were overly complex or designed with the needs of professional software engineers in mind. Therefore, the **adaption** of existing tools could help solve these problems. Finally, participants discussed a lack of **incentive**: their job was not set up to incentivize adherence to these best practices we identify. To solve this, relevant decisions, including those involving tenure and journal article acceptance could integrate the contribution of code, which adheres to these best practices. One way to do this would be to mandate that authors submit code to accompany the associated journal article it was used for, so that the code could be judged in the existing peer review framework.

Participants’ development processes were creative and extremely varied, such that determining whether a participant's activities were in accordance with a given best practice required the use of considerable subjective interpretive judgment. More work in the way of defining best practices in concrete granular detail, within the scope of the scientific process and in collaboration with scientific coders, would go a long way towards removing this subjectivity.

Appendix

Pre-Interview Questionnaire

1. What is your field of research?
2. What is your job title?
3. What and in which field is your highest degree? (eg: MS Computer Science, or PhD Physics)
4. For what percent of your time at work do you write code?
5. How many other people work on the same code project at your work? (enter 0 if solo project)
6. If other people work on the same code project, do you lead the project?
7. How did you learn to write code? (check as many as apply, feel free to describe other methods in the other box)
8. Which programming languages do you use in your research? (choose up to 3)
9. What is the purpose of the code you write at work? Feel free to list many. (eg: end user software, machine learning algorithms, robot control code, database code, web design)
10. Do you ever write code for personal or non work related purposes?
11. Is your system based on code from someone else? If so, where did it come from?
12. How did you determine the goals you wanted your system to satisfy?
13. What did you do to design your system before you began writing actual code?
14. How do you keep track of different versions of your system?
15. What process do you use to test your code?
16. What software or tools do you use to write your code?
17. What process do you use to update or maintain your code?
18. Do you release your code publicly? If so, how do you package and release it?

Interview Script

1. What does your code do? What does it allow you to do that you couldn't do without writing code? What created the need?

2. How did your code project start? What were the first steps you took to fill this need?

3. What are the three hardest parts about building or maintaining your code?

4. What tools do you use to build your code (list all)? Why did you choose these tools? What do they do well? Where do they fall short?

5. If you had several years of experience as a software engineer how do you think this would change the way you build code? If you had a CS degree?

6. What resources have you used to learn how to build code? If you were to coach someone else in your discipline on how to learn to code, what advice would you give them? What would you tell them to do?

7. What role will code play in future advances in your field? Do you think that coding will soon become a required skill in order to conduct high quality research?

8. Additional questions and follow ups as needed.

Bibliography

Adrion, W. Richards, Martha A. Branstad, and John C. Cherniavsky. "Validation, verification, and testing of computer software." ACM Computing Surveys (CSUR) 14.2 (1982): 159-192.

Barnes, Nick. "Publish your computer code: it is good enough." Nature 467.7317 (2010): 753.

Baxter, Susan M., et al. "Scientific software development is not an oxymoron." PLoS Comput Biol 2.9 (2006): e87.

Carey, Stephen S. A beginner's guide to scientific method. Cengage Learning, 2011.

Carver, Jeffrey C., et al. "Software development environments for scientific and engineering software: A series of case studies." Software Engineering, 2007. ICSE 2007. 29th International Conference on. Ieee, 2007.

Guo, Philip J., and Margo Seltzer. "BURRITO: Wrapping Your Lab Notebook in Computational Infrastructure." TaPP 12 (2012): 7-7.

Guo, Philip Jia. Software tools to facilitate research programming. Diss. Stanford University, 2012.

Hames, Irene. Peer review and manuscript management in scientific journals: guidelines for good practice. John Wiley & Sons, 2008.

Morling, Beth. Research methods in psychology: Evaluating a world of information. WW Norton & Company, 2014.

Myers, Brad A., Andrew J. Ko, and Margaret M. Burnett. "Invited research overview: end-user programming." CHI'06 extended abstracts on Human factors in computing systems. ACM, 2006.

Myers, Brad A., Andrew J. Ko, and Margaret M. Burnett. "Invited research overview: end-user programming." CHI'06 extended abstracts on Human factors in computing systems. ACM, 2006.

National Academy of Sciences, National Academy of Engineering, and Institute of Medicine. 1992. Responsible Science, Volume I: Ensuring the Integrity of the Research Process. Washington, DC: The National Academies Press. doi:https://doi.org/10.17226/1864.

Ruparelia, Nayan B. "Software development lifecycle models." ACM SIGSOFT Software Engineering Notes 35.3 (2010): 8-13.

Ruparelia, Nayan B. "Software development lifecycle models." ACM SIGSOFT Software Engineering Notes 35.3 (2010): 8-13.

Segal, Judith. "Some problems of professional end user developers." Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on. IEEE, 2007.