Software Carpentry Assessment Report

Jorge Aranda

July 4th, 2012

Contents

1 Executive Summary **2**

2 Introduction and Methodology **3**

3 Analysis of Software Carpentry's Impact **6**

3.1 Survey data 6

3.2 Interview and observation data 11

3.3 Summary of findings 19

4 Recommendations **20**

4.1 Workshop improvement 20

4.2 Subsequent assessments 21

A Survey template **23**

B Interview scripts **27**

Chapter 1

Executive Summary

This report summarizes a six-month effort to assess the efficacy of the Software Carpentry program. Through a mixed-methods approach, including surveys, pre- and post-workshop interviews, workshop observations, and screencast analysis, this assessment concludes that the key premises for the usefulness of Software Carpentry instruction hold true: most scientists are self-taught programmers, they have fundamental weaknesses in their software development expertise, and these weaknesses affect their ability to answer their research questions.

More importantly, this assessment concludes that Software Carpentry instruction helps scientists eliminate these weaknesses. The program increases participants' computational understanding, as measured by more than a two-fold (130%) improvement in test scores after the workshop. The program also enhances their habits and routines, and leads them to adopt tools and techniques that are considered standard practice in the software industry. As a result, participants express extremely high levels of satisfaction with their involvement in Software Carpentry (85% learned what they hoped to learn; 95% would recommend the workshop to others).

While the outcome is largely positive, there are areas for improvement. Two of note are the spread in expertise among participants and the barriers that they face to change their software practice. The first problem leads to some participants feeling that the instruction is too advanced or too basic for them. The second problem damages the impact of Software Carpentry instruction, as participants learn valuable material, but find that for other reasons they are unable to adopt the relevant tools and techniques. Both problems, and other minor issues identified, can be addressed by the Software Carpentry team in subsequent versions of their workshop.

Chapter 2

Introduction and Methodology

The mission of the Software Carpentry program is to help scientists be more productive by teaching them basic computing skills. Software Carpentry aims to achieve this through short, intensive workshops (bootcamps) and self-paced online instruction.

Over the years in which Software Carpentry has been offered, informal feedback, anecdotal experience, and word of mouth enthusiasm, have suggested that scientists benefit from the program, but there has not been an effort directed specifically to assess its impact, nor its underlying assumptions about the computing needs of scientists. From January to June 2012, I was engaged to carry out two tasks: to conduct an assessment of the program, using a mixed-methods approach, in order to evaluate its impact on workshop participants and its areas for improvement, and to construct a repeatable, systematic, and efficient assessment strategy that can be used in subsequent workshops. This document presents the result of the work in those two fronts.

This assessment of the Software Carpentry program had two main goals:

1. To inform the Software Carpentry team whether the instruction has the desired impact, and the areas in which the program could be improved for future iterations of the workshop.

2. To formulate a long-term repeatable, systematic, and sustainable evaluation process to be applied by researchers and workshop organizers in the future.

In order to satisfy these goals, the assessment project used a combination of quantitative and qualitative data collection and analysis, consisting of the following:

- Pre- and post-workshop surveys of participants from eight different workshops in Canada, the United States, and the United Kingdom (a survey of a ninth workshop is still under way).
- Interviews of participants from five different workshops in Canada and the United States.
- Fly-on-the-wall observations of one workshop.
- Analysis of screencasts recorded by workshop participants as they worked through a programming assignment.

In parallel, Professor Julie Libarkin from Michigan State University performed a detailed assessment of participants in the workshop held there (which was also attended remotely by students from the University of Texas at Austin). Professor Libarkin's report is independent from this work; nevertheless some of her findings are mentioned in this report where appropriate.

The pre- and post-workshop surveys were conducted online, using LimeService[1] as the hosting service. A pilot of the survey was tested with University of British Columbia students, and later refined. Participants from a total of eight workshops participated in the survey.[2] Pre-workshop survey invitations were sent roughly one week before the start of the corresponding workshop; post-workshop invitations were sent between three weeks and three months after completion of the workshop, in order to give participants enough time to digest the material and for any effects of the workshop to begin establishing in their daily lives. Participation was voluntary but not anonymous.

The survey asked questions regarding the software development habits of respondents, the tools they were familiar with, their level of knowledge of five core Software Carpentry topics (shell commands, Python, version control, SQL, and testing

[1] http://www.limeservice.com/

[2] These were held at Indiana University, Michigan State University, the Monterey Bay Aquarium Research Institute, NERSC, Utah State University, the University of Alberta, the University of British Columbia, and the University of Newcastle.

concepts), and their challenges in using scientific computing to answer their research questions. Appendix A presents the full survey template. We collected 278 responses to the survey in total.

Concurrently, we interviewed participants from five workshops.[3] There were pre-workshop interviews, interviews during one workshop, and post-workshop interviews. Interviews lasted between fifteen minutes and one hour; there was a total of 69 interviews. All interviews except the more informal ones (those carried out during a workshop) were semi-structured, and either recorded or annotated, and later analyzed to examine patterns, similarities, differences, and improvement proposals offered by participants. Appendix B shows the standard questions asked during interviews; note that in practice interviews frequently deviated from this script as the situation demanded.

Workshop observations took place at the National Energy Research Scientific Computing Center (NERSC) at Berkeley, California, in March 28th and 29th. While it was possible to conduct some informal interviews during the workshop, the aim of the visit was to observe the workshop instruction and the reactions from participants. These observations offered an opportunity to understand the level of what is feasible during a two-day window, the background of workshop participants, their expectations, and their problems as they attempt to follow the instruction.

Finally, we performed an analysis of seven screencast recordings made by workshop participants as they worked through a programming assignment. Participants were asked to think aloud as they worked through their assignment, and to avoid editing their screencasts in order to capture a more accurate picture of their thought process and challenges. The goal of these screencast analyses was to evaluate the prospect of using a similar mechanism as part of Software Carpentry's assessment efforts; we discuss their potential in section 4.2.

[3] At NERSC, the Space Telescope Science Institute, the University of Alberta, and the University of British Columbia (twice).

Chapter 3

Analysis of Software Carpentry's Impact

3.1 Survey data

We had a total of 278 responses to our survey (an average of 35 per site). Of these, 191 were pre-workshop responses, and 87 were post-workshop responses (respectively, an average of 24 and 11 per site). Since we asked for respondents' names, we were able to match the pre- and post-workshop data of 71 respondents. Since this subset of our data provides both pre- and post-workshop responses from the same people, it is of particular importance for our analysis. In the rest of the discussion, we make a distinction between figures from the "full set" (or FS) and the "paired set" (or PS) of responses; the latter corresponding to the subset of 71 respondents described above.

First, we analyzed the level of use of Software Carpentry tools and techniques. Respondents rated their "level of use" of these tools and techniques, before and after the workshop, in a scale with values Do Not Use, Sometimes, and Frequently. We converted each of these levels into an ordinal and compared the difference before and after the workshop; numerically, an increase of one unit is equivalent to a jump of one level. The level of use of *all* tools and concepts, with the exception of SQL, increased after the workshop. The particular figures for all tools and techniques are presented in Table 3.1.

The "quiz" items consisted of Yes/No questions, four per topic, that were purpose-

Response set	Shell commands	Python	Testing	Version control	SQL
Full set	+0.32	+0.21	+0.34	+0.31	-0.04
Paired set	+0.29	+0.33	+0.34	+0.26	-0.01

Table 3.1: Change in the level of use of Software Carpentry topics

fully chosen so that only about *half* of them could be answered with the standard material in the workshop; the other half was not covered by workshop instruction. Additionally, a cross-cutting half could be answered with introductory familiarity to the topic in question, while the other half would represent more advanced levels of expertise. In other words, the quiz was designed so that there were an easy and a hard question answerable through workshop instruction, and an easy and a hard question not answerable through workshop instruction.[1] The objectives for this were to assess whether participants not only learned the workshop materials, but were exploring the topics in greater depth on their own, and to avoid ceiling effects in our survey.

Quiz performance improved across the board for all question categories, both for the full set and for the paired set. Table 3.2 shows the mean performance of respondents on the quiz questions, out of a score of 100 marks. As can be seen by these numbers, improvement was considerable and fairly uniform, with the exception of testing concepts, which improved less than the others.

To analyze the "needed for work" questions, in which respondents rate the relevance of several core Software Carpentry tools and concepts in a scale with values I Don't Know What This Is, Unimportant, Marginal, and Important, we converted each of these levels into numerical values (where the first two categories equal zero, Marginal equals one, and Important equals two), and compared the difference before and after the workshop. The relevance for our respondents' work of all tools and concepts increased, albeit in the case of SQL and Python the increase was negligible for one of our two sets. The particular figures for all tools and techniques are presented in Table 3.3.

More people had an online presence in GitHub, BitBucket, or StackOverflow after the workshop than before (+36.9% more people in the full set, +36.8% in the paired set).

[1] In practice this was hard to achieve, given that different workshops approached topics in a slightly different manner, partly due to participants' questions.

Response set	Before	After	Difference
Shell commands			
Full set	45.3	75.2	+29.9
Paired set	43.7	76.1	+32.4
Python			
Full set	33.7	63.4	+29.7
Paired set	30.3	65.4	+35.1
Testing			
Full set	11.7	23.8	+12.1
Paired set	9.5	24.6	+15.1
Version control			
Full set	15.2	46.4	+31.2
Paired set	10.2	44.5	+34.3
SQL			
Full set	23.4	48.2	+24.8
Paired set	19.0	49.6	+30.6

Table 3.2: Mean quiz performance

Response set	Shell commands	Python	Testing	Version control	SQL
Full set	+0.15	+0.03	+0.43	+0.44	+0.01
Paired set	+0.07	+0.20	+0.36	+0.41	+0.14

Table 3.3: Change in the relevance of Software Carpentry tools and techniques in the work of its participants

Response set	Before	After
Full set	46.99	45.60
Paired set	46.33	46.17

Table 3.4: Mean work hours per week

This may point to an increased engagement with communities of developers around the world, and to a greater adoption of version control tools and environments.

As Table 3.4 shows, the mean number of work hours was practically the same before and after the workshop. Table 3.5 presents the figures for work hours spent working on creating or modifying software; they remained stable with a slight increasing trend.

Informal assessments of hours gained or lost were for the most part uncertain. Many people said it was still too early to tell how many hours were gained or lost by participating in the workshop. Among the few that claim a loss (8 from 87 post-workshop responses), the reasons are longer-term payoff (that is, that attending the workshop will probably pay off, but it still has not, usually because the workshop happened too recently), or further learning investments (that is, that they are still putting in time to learn some computational tools more deeply or installing infrastructure, and that larger investment has not yet finished paying off). Among those that already claim

Response set	Before	After
Full set	11.70	13.24
Paired set	11.36	11.83

Table 3.5: Mean work hours per week spent working on software

a gain (15 out of 87 post-workshop responses), the number of hours tends to fall in the 1hr-5hr weekly range.

Overall, 13 out of 87 (15%) post-workshop respondents (approximately; the survey responses are a bit subjective in this regard) point to new research questions or new computational approaches that they are able to take on thanks to the Software Carpentry workshop. A larger number says that it is too soon to answer whether the workshop has opened new research horizons for them.

Survey respondents report a wide variety of habit changes and other benefits from taking the workshop. Roughly, these fall on the following categories:

- Proficiency with the tools covered in the workshop (shell, version control, Python, SQL).
- Regular use of computing tools (shell scripting, version control).
- Greater concern with scientific computing issues (provenance, more rigorous testing).
- Improvements in the respondent's approach to develop software (efficiency in programming, control in length of programming sessions, test-driven development, automated testing, better requirements/problem statement definition, code structure).
- Improvements in the respondent's outlook towards developing software (better code comprehension, higher confidence in one's own ability to program, better perspective on roadmap for learning to design software).

Survey respondents on the paired set have interesting changes regarding research goals they cannot or could not attain due to a lack of computational skill.[2] For 11 respondents, the original challenge remains in some form. For 7 respondents, the old challenges disappeared, and new challenges are mentioned. For 10 respondents, the new challenge description appears to be a refinement over the old, with a clearer computational statement of what is needed to tackle the challenge. Finally, 20 respondents reported a challenge before the workshop, but the challenge disappeared after the workshop. This represents a total of 48 paired responses for which this analysis was viable, out of 71 paired respondents.

[2] Note that analysis of these is necessarily subjective.

The "reasons for inaction" responses were not amenable to analysis. They were too scattered, inconsistent before and after the workshop, or besides the point actually being asked. This question, at least in its present form, should be dropped in further evaluations.

Professor Libarkin's own study provides additional support for these findings. Her post-workshop survey was administered immediately after the end of the workshop, so she could not explore the adoption of new habits nor the shift in research questions that we probed in our survey. She reports the following:

- Prior to instruction, participants expressed either no or low ability in computational ability (1.73 ± 0.49; where 1 represents No Ability and 4 represents High Ability).
- Participants perceived greater computational understanding after engagement in the workshop (an increase from 2.26 to 2.91, where 1 implies Low Understanding and 4 implies High Understanding).
- Participants perceived greater Python coding ability after engagement in the workshop (from 2.48 to 3.00, same scale as above).
- Participants were generally very satisfied with the workshop (81% felt their computational understanding changed; 85% felt they learned what they hoped to learn).
- Participants generally felt the workshop met their needs and would overwhelmingly (95%) recommend it to others.

In every point, these findings lend further support to the findings in this report.

3.2 Interview and observation data

There were 69 interviews in total (38 pre-workshop, 12 during a workshop, and 19 post-workshop). These interviews, along with the workshop observation carried out at NERSC, allowed us to determine our interviewees' abilities and grasp of scientific computing issues before and after the workshop.

Overall, interview data confirm several key assumptions of the Software Carpentry program:

A large portion of scientific computing researchers are self-taught programmers. Typically, participants have had between zero and two introductory courses to Computer Science or programming, and their knowledge of programming arises from their practical research needs. In fact, as the experience of many interviewees shows, to date one can still have a successful research career, in which one quarter of one's time (or more) goes into creating and modifying software, and yet have *never* worked, been advised by, been trained by, or been in even infrequent consultation with professional software developers.

Scientific computing researchers are not familiar with basic software development tools and techniques. Workshop participants may have a sophisticated control of a particular tool, either generic or specialized for their particular niche, but that does not entail expertise in software development in general. It was common to find participants that, before the workshop intervention, had never heard of tools that professional developers take for granted, such as the shell, version control, or SQL. Among those familiar with the terms, it was also common to find researchers who did not see the point in using them.

In this sense, software testing was most problematic. It was rare to find participants who had any kind of automated or regression tests for their code. Reasons for not testing abound—the typical ones are that the scripts produced are too small to have errors, that one does not know their answers in advance and therefore cannot automate the testing process, and that given the nature of the code produced (simulations looping a great number of times) any errors in the code are immediately apparent through simple inspection.

These shortcomings reduce scientists' ability to answer their research questions. One interviewee, for example, lamented that nobody in his lab (including himself) had the computing expertise needed to troubleshoot software development issues; he would reportedly spend weeks solving issues that he felt should have been trivial for a competent software developer. Another explained that he was sure he was extremely inefficient in his programming, but that never having seen a professional developer at work, he did not know how expertise looked like, and what would be a good way to focus his learning efforts. Several scientists pointed out that the reason why they registered to the Software Carpentry workshop is because their field is becoming increasingly computational, and they feel they are being left behind.

Researchers that are less aware of their computational weaknesses are also affected by them, but they are generally unable to describe their missed potential. When discussing computational problems they describe issues that are easily tackled through

Software Carpentry tools; sometimes they describe a problematic situation without realizing that it is problematic.

Attendance to a Software Carpentry workshop helps eliminate these weaknesses. The survey responses described in Section 3.1 point to several benefits of the Software Carpentry intervention, and the collected interview data confirm these benefits. Software Carpentry participants claim that the workshop has helped them identify their computational weaknesses, that through direct instruction it has contributed to mitigate those weaknesses, that it has shown them where and how can they continue to learn more, and that it has enabled them to effect larger changes in their labs and working groups.

One of the topics that found the most traction among scientists is version control. Before the workshops, version control tools were generally dismissed by interviewees as not applicable to their case or too much of a bother. After the workshops, many interviewees, as well as their research labs or groups, either got a version control repository installed and in use, or are planning to do so for their next project. To a lesser extent this was also true for the use of Python, of shell commands and shell scripts, and of testing practices.

Interviewees also claim to have improved their assessment of their computing skills, and to have gotten practical knowledge on how to improve them. Interviewees state that their code is better structured now, that they are better able to control the duration of their programming sessions, and to work in shorter cycles. In general, the benefits listed by participants during interviews are similar to those found in the surveys.

The interviews and observations also helped us identify strengths and weaknesses with the current version of the workshops and online instruction. Several of the most significant findings in this regard are the following:

Enthusiasm for Software Carpentry. Participants were overwhelmingly enthusiastic about the Software Carpentry program, the Software Carpentry team, and the pedagogical approach, and they want more Software Carpentry engagements in their venues. They claimed that they were extremely glad to have taken the workshop, that they wished to have taken it years before, and that they would recommend it to other people. Two interviewees said that they were fascinated that the workshop was available for free. Not a single interviewee (there were 19 post-workshop interviews) deviated from this trend.

Participants were also enthusiastic about the instructors and their pedagogical style.

The importance of an appropriate instructor was brought up several times: the workshop covers much ground in a short time, and for interviewees it was a relief to have instructors that made the material entertaining, compelling, and useful at the same time.

Barriers to adopt Software Carpentry tools and techniques. In many cases, workshop participants faced significant barriers to adopt Software Carpentry tools and techniques in their daily work, and by the time of their interviews they had not yet surmounted them, would not know how to do so, or judged that trying was not cost-effective.

These barriers are of three kinds: infrastructural, organizational, and habit-based. Infrastructural barriers consist of technical problems that cannot be solved without more resources or greater skills than the interviewee currently possesses. For example, the participant may be convinced of the value of version control, but she has no access to a system administrator, does not know how to set up a server, and everything that she reads online about how to do it confuses her further.

Organizational barriers consist of mismatches between the new-found conviction of workshop participants and the dynamics in their labs. The participant may be convinced, again, of the value of version control, but none of their peers is, and she must abandon her attempt to institute a version control-based collaboration.

Habit-based barriers consist of the ingrained habits of workshop participants themselves: the challenge is not external (neither technical or organizational), but internal. For example, the participant is convinced of the value of proper testing, but she is not used to doing so, and for her next scripts, which she needs to complete as quickly as those she has written in the past, she does what she always does and what has been moderately successful in the past, and she again forgets to test as she meant to do.

These barriers to adoption are not universal (some participants changed their work practices significantly since the workshop), but they are widespread. Efforts to counter them should be part of the next iteration of Software Carpentry workshops.

Variety of participants and instruction pace. Bootcamps face the challenge of finding an adequate instruction pace, as participants have widely varying levels of expertise. For each of the workshop components there are novices and veterans, and this is problematic for instruction: the instructor wants to bring everyone ahead, but aiming for the average participants means that many will still be left behind and many will find the content too basic. Problems of skill variation were brought up

several times by interviewees.

Most other characteristics of participants also vary widely. Participants come from different domains, have different research goals, different roles, and needs. Reaching all of them is difficult.

"Strategic" lessons about development practice. The Software Carpentry program attempts to teach several principles for scientific computing, and to transmit a number of recommendations for computational scientists.[3] However, while this goal is not covert, it is not usually broadcasted to potential workshop participants, under the (correct, as interview data show) assumption that many potential participants do not know that they need deeper lessons about computing, but they do know that they need to learn a specific tool or language.

The Software Carpentry website lists eleven principles and eleven recommendations, and the observations at the NERSC workshop allow us to examine the extent to which transmitting these principles and recommendations along with the technical content is feasible in the limited time available. The following list presents all the Software Carpentry principles and discusses their presence or absence at the NERSC workshop:

1. *Code is just a kind of data.* Discussed briefly, under data and code provenance.
2. *Metadata makes data easier to work with.* Discussed briefly, under data and code provenance.
3. *Separate models and views.* Not discussed.
4. *Trade human time for machine time and vice versa.* Discussed, in the context that it is important to figure out what would be faster: to program something or to do it oneself; and that the answer depends on the speed of doing it and the number of times one is expected to do it.
5. *Anything that's repeated will eventually be wrong somewhere.* Discussed repeatedly, almost verbatim.
6. *Programming is about creating and composing abstractions.* Discussed, in the context of function creation.

[3] http://software-carpentry.org/2012/01/the-big-picture-2/

7. *Programming is about feedback loops at different timescales.* Discussed, when talking about the state of flow, small tasks, and large tasks, although the emphasis was on the timescales, not on the feedback loops.

8. *Good programs are the result of making good techniques a habit.* Discussed at length. Thinking about habits was central to the instruction at the workshop. For example, there were repeated mentions of a version control habit (update, merge, edit, commit), and of the test-before-code habit.

9. *Let the computer decide what to do and when.* Not discussed.

10. *Sometimes you copy, sometimes you share.* Discussed briefly, in the context of Open Science.

11. *Paranoia makes us productive.* Discussed, in the context of test-driven development.

Similarly, the following list shows which recommendations were explicitly formulated to workshop participants at NERSC:

1. *Use the right algorithms and data structures.* Not discussed.

2. *Use a version control system.* Discussed at length.

3. *Automate repetitive tasks.* Discussed in several contexts, for instance, when the instructor made a point about saving a list of likely repeatable shell commands and putting them in a script.

4. *Use a command shell.* Discussed at length.

5. *Use tests to define correctness.* Discussed at length.

6. *Reuse existing code.* Discussed, especially in the context about the flexibility and versatility of the "piping with small scripts" model.

7. *Design code to be testable.* Discussed briefly, in the context of test-driven development.

8. *Use structured data and machine-readable metadata.* Discussed in the context of databases and SQL.

9. *Separate interfaces from implementations.* Not discussed.

10. *Use a debugger.* Discussed at length.

11. *Design code for people to read.* Discussed, in the context of structuring code for readability and providing the right level and kind of comments.

As these lists show, a majority of the principles and recommendations found their way into the 2-day NERSC workshop. Some of those that did not are an awkward fit for the core Software Carpentry content, and it may only be feasible to incorporate them if the core workshop materials are modified.

Experienced participants pointed to these lessons as one of the reasons for them to attend the workshop; they were also one of the most uniformly mentioned benefits participants reported afterwards. Participants use several labels to describe these strategic lessons ("programming hygiene", "learning about the rhythms of software development", "philosophy of programming", and so forth), referring to the same push of Software Carpentry to instill better habits and working practices in software developers.

Seeing an expert at work. In a similar vein as the previous point, one of the most cited observations from post-workshop interviews was that they learned much from seeing an experienced developer (the instructor) work through problems live and thinking aloud. They claimed that this allowed them to better understand issues of programming practice and code structure. Interviewees reported several benefits associated with this instruction style: they claim they were able to reason about their code better, to structure their code more appropriately, to incorporate testing practices where they had not before, and to automate tasks. The interviews did not probe the question of whether participants felt they would get similar benefits from watching a recorded video or the expert at work.

Other benefits. Unprompted, several interviewees claimed to have reaped other benefits from the workshop: the formation of local communities of like-minded enthusiasts, partly through online sessions; an improvement in self-confidence with respect to computing skills, as the workshop demystified some aspect of computing that they were now ready to deal with in their own terms; and knowledge of sources where they could look for more information, which helped them deal with the overabundance of esoteric technical information available on the Internet.

Effective argumentation. In post-workshop interviews, participants mentioned three kinds of arguments as compelling them to lend credence to the principles and rec-

ommendations advocated by Software Carpentry. First, discussions of findings from empirical software engineering. This is in contrast with the rejection to these findings that professional software developers often have, under claims that empirical research does not accurately capture the complexities of their field. Second, learning that a tool or technique is part of the toolset or a habit of professional software developers. This is, in part, what many workshop attendees expected to find from the workshop, beyond the technical instruction. Third, an explanation of the underlying rationale for using the tools and techniques advocated by Software Carpentry. These explanations, in the NERSC workshop, were often presented through a story or anecdote, and interviewees stated that this approach worked for them.

Resources in the Software Carpentry website. Reactions to the materials in the Software Carpentry website were varied, but mostly positive. At one end of the spectrum, respondents said that the website did not give them information at the level or in the format they wanted it. This was particularly true of more experienced participants. At the other end, some participants found the material in the website extremely useful. Opinion was polarized on whether the videos or the screenshot/transcript combination was better; in any case, these participants benefitted from the website and recommended it to their peers.

Revisions to the material. Throughout the post-workshop interviews, respondents raised issues with three of the core Software Carpentry components. First, regarding the Python material, some of the more knowledgeable Python programmers felt that the level of instruction did not do justice to its scientific computing capabilities; that if their peers were to judge the merits of Python for scientific computing on the basis of what was presented in the workshop, the verdict would be unsatisfactory.

Second, many participants did not find the SQL material useful or well connected to their computational challenges. It is unclear if this disconnection is self-imposed, or if SQL truly is not an appropriate solution for many scientific computing challenges; in any case it was perceived as such.

Finally, the version control tool of choice (Subversion) is falling out of fashion in this community, and several participants expressed that they would have preferred to learn about a distributed version control system instead, Git being mentioned most frequently.

3.3 Summary of findings

The aggregate of the survey data, as well as the interview and observation data, strongly suggest that Software Carpentry workshop participants are satisfied with the program and that they get considerable benefits from attending. The workshop helps them learn practical skills, and it also teaches them less technical but more relevant lessons about computational thinking and proper software development habits.

The collected data also point to areas of improvement. Participant expertise is widely spread, and therefore the instruction pace is necessarily problematic for some attendees. Some core topics need to be revised to increase their relevance for the majority of workshop participants. Finally, the Software Carpentry program needs to evaluate how to better help participants overcome the barriers to adoption of core tools and techniques, in order to have an even larger impact in their research potential.

Chapter 4

Recommendations

The participants of the Software Carpentry program found the workshop beneficial and useful, and all available data point to participant learning and improvement in computational skills. Nevertheless, I collected several recommendations that I think should be considered for future iterations of the workshop, and for further assessment efforts.

4.1 Workshop improvement

Based on the survey, interview, observation, and screencast data collected in these past six months, I believe the Software Carpentry team should consider the following suggestions for improvement.

1. **Divide workshop participants by level of expertise.** One of the most common suggestions for improvement that arose from the interview data is to have participants self-select into a group of novice or advanced attendees. Self-selection is problematic, as many participants will not be accurate judges of their own expertise; one approach is to present participants with a brief but concrete list of commands and concepts that they would be expected to be familiar with in the advanced workshop.

2. **Assist participants in overcoming barriers to adoption.** Several participants struggled not with the course instruction, but with implementing the changes

that Software Carpentry advocates for in their daily lives. A mentorship model during the online sessions can be used to help them overcome these barriers.

3. **Revisit workshop materials based on scientists' needs.** While most of the material is current and relevant, three recommendations for revision arose from this evaluation. First, the Python material does not delve into the real advantages of Python as a tool for scientific computing. Second, the SQL material is perceived as less relevant by many participants. Third, the version control tool of choice (Subversion) is not as popular as Git or Mercurial anymore, and participants generally would like to start by learning the latter instead of Subversion.

4. **Increase hands-on and peer-led learning.** Participants expressed an interest in doing more exercises during the bootcamps; the time allocated to do so in the NERSC workshop that I observed was very limited. Some participants also wanted more peer-led efforts, which can be cultivated via the post-workshop online sessions.

5. **Address installation and technical issues.** One time waster during the workshops is installing cygwin for participants that come with Windows machines. Despite pre-workshop warnings, several come without having installed cygwin; even when they have, they often miss the packages that they will need afterwards. After the workshops, holding online sessions remains notoriously riddled with technical issues, enough to turn people away from participation.

6. **Revisit homework assignment design and evaluation.** Homework assignments for the online sessions are useful for participants, but two modifications would increase their value. First, whenever possible, they should be better integrated with the particular kinds of problems and computational challenges that participants face. Second, participants would appreciate getting more (and more timely) feedback on the assignment code they produce, in order to improve upon their practice.

4.2 Subsequent assessments

Performing this assessment was a fairly time-intensive project. It involved travel to four locations and weeks of data collection and analysis. One of its goals, however,

was to formulate a better (more efficient, replicable, sustainable) strategy for subsequent assessments. I recommend that subsequent program assessments adhere to the following strategy:

1. **Continue administering pre- and post-workshop online surveys.** Maintain the current pre- and post-workshop online survey, with modifications for workshop content, with the addition of a question about workshop satisfaction, and with the elimination of the least informative questions (on "reasons for inaction" and on "participation in online development communities"). Interview data largely confirmed the findings from the survey, suggesting that the survey can be used as the main evaluation instrument for the program.

2. **Include a random sample of screencasts in the analysis.** Have instructors watch over a random selection of screencasts produced by participants after their workshop, where they address a programming task. The goal of screencast analysis would not be to grade learning in the workshop, but to help instructors understand the common obstacles and misunderstandings that their participants face. Screencast analysis need not be systematic nor formalized to have this effect.

3. **Use interviews to validate survey data and assess satisfaction.** Continue to validate the usefulness of the survey by performing a limited amount of short post-workshop interviews, conducted over the phone, equivalent to about half a day of work, including set-up, data collection and analysis (that is, about four to six interviews). The goal of these interviews should be to gauge participant satisfaction and to identify whether Software Carpentry's efforts at mitigating the problems uncovered so far are being successful or not.

Altogether, these efforts should require approximately two days of work per workshop, and they need not be exerted for every workshop. They can all be carried out by the instructor or by one of their helpers; the post-workshop questions in the interview script in Appendix B should yield useful information to the interviewer, and can be used as the basis for subsequent interviews.

Appendix A

Survey template

The following text and questions was presented to all survey respondents, with formatting differences to account for LimeQuery's capabilities. All questions but the first were optional.

Software Carpentry evaluation survey — *Name of Venue*

Thanks for helping us evaluate the Software Carpentry project. This survey should take you 10-15 minutes to respond. While you provide your answers, please remember that you are not being graded! It will be most useful for us if you try and give your most honest assessment on the following questions.

What is your full name? (We'll apply this survey twice, and we need your name to compare your answers between both applications)

What is your affiliation?

In a typical week, about how many hours do you work? (If you are a student, your studies count as work.)

About how many of those hours do you spend creating, modifying, or testing software?

Indicate your level of use of each of the following tools or techniques: *Likert scale with three values (Do Not Use, Sometimes, Frequently) plus a No Answer option.*

- Shell commands
- Testing

- SQL
- Version control
- Python

Do you understand the following shell commands well enough to explain them to somebody else? *Options: Yes, No, No answer.*

- `ls data/*.txt`
- `sort elements.txt > elements.txt`
- `find ~-name `*.py'`
- `ps -A | grep mysample`

Do you understand the following Subversion commands well enough to explain them to somebody else? *Options: Yes, No, No answer.*

- `svn update`
- `svn merge -r 42:45 ../pathToTrunk`
- `svn move foo.py bar.py`
- `svn diff -r 1234`

Do you understand the following Python commands well enough to explain them to somebody else? *Options: Yes, No, No answer.*

- `x = {`east' : 5, `west' : 11}`
- `[x**2 for x in aList]`
- `for i in range(len(aList)):`
- `__init__(self, *args)`

Do you understand the following testing concepts well enough to explain them to somebody else? *Options: Yes, No, No answer.*

- assertion
- exception
- fixture
- mock object

Do you understand the following SQL commands and concepts well enough to explain them to somebody else? *Options: Yes, No, No answer.*

- `select * from data where data.left < data.right;`
- foreign key
- `drop table "logs"`
- inner join

How important is each of the following items in your work? *Likert scale with four values (I don't know what this is, Unimportant, Marginal, Important) plus a No Answer option.*

- Shell commands
- Testing
- SQL
- Version control
- Python

Do you have research goals that you cannot attain because of a lack of computational or programming expertise? If so, please elaborate.

If some of the tools or techniques above could make your work easier, but you have not learned them in depth, please tell us why.

Have you completed a Software Carpentry workshop or bootcamp? *Possible answers: Yes, No, No answer. If the answer is Yes, the following three questions become available.*

Estimate how many hours have you gained or lost **weekly** as a result of the workshop. Please elaborate.

Can you think of computer-related things that you are doing differently than you used to since taking the workshop? If so, please elaborate.

What scientific results have you produced (or are you producing) that would have been difficult, impossible, or out of reach before this training?

Are you registered on GitHub, BitBucket, and/or StackOverflow? *Possible answers: Yes, No, No answer. If the answer is Yes, the following question becomes available.*

For those services for which you have registered (GitHub, BitBucket, and StackOverflow), please give us your usernames.

Appendix B

Interview scripts

The following script was used for pre-workshop interviews:

- What is your position? What are your main responsibilities?
- Do you develop or modify software? How frequently? For how long? For what purpose?
- Do you use the shell? How comfortable are you using it? Examples?
- Do you use Python? How comfortable are you using it? Examples? How about other programming languages? How did you learn to use them?
- How do you manage the versions of your code and data? Do you use version control software? For what purpose? How?
- Do you use databases?
- What other tools do you use? IDEs? What operating systems are you familiar with?
- What can you tell me about your testing practices? Do you have automated tests? How do you assess if your software behaves appropriately?
- Are you concerned about data or code provenance issues? What do you do about it?

- If you were explain to others how good software development looks like, what would you say?
- What are you expecting to get from the workshop?

The post-workshop interviews covered the same ground, except for the last question, which was substituted for the following set of questions:

- What were your impressions of the workshop? Both positives and negatives.
- Has anything changed in your approach to computing or software development since you took the workshop? Please walk me through an example.
- Have your routines changed? In what ways?
- Did any of the more philosophical or strategic points that the instructor made stick with you? If so, can you give me some examples?
- Are you tackling the same research questions? Have they evolved?
- Anything else you'd like to say?

Naturally, for those interviewees that were interviewed twice (before and after the workshop), the background questions were not repeated.